AD-A254 311

ON PAGE

Form Approved
OMB No. 0704-0188

1

1. AG... ONLY (Leave blank)	2. REPORT DATE August 1992	3. REPORT TYPE AND DATES COVERED THESIS

4. TITLE AND SUBTITLE An Interactive Artificial Cutting Plane Method for Bicriterion Integer Programming Problems	5. FUNDING NUMBERS
6. AUTHOR(S) Diane Breivik Allen, 1st Lt	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) AFIT Student Attending: Mississippi State University	8. PERFORMING ORGANIZATION REPORT NUMBER AFIT/CI/CIA-92-066
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) AFIT/CI Wright-Patterson AFB OH 45433-6583	10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for Public Release IAW 190-1 Distributed Unlimited ERNEST A. HAYGOOD, Captain, USAF Executive Officer	12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

DTIC
ELECTE
AUG 25 1992
S B D

92-23499

92 8 24 008

14. SUBJECT TERMS	15. NUMBER OF PAGES 80
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT

AN INTERACTIVE ARTIFICIAL CUTTING PLANE METHOD FOR BICRITERION INTEGER PROGRAMMING PROBLEMS

By

Diane Breivik Allen

A Thesis
Submitted to the Faculty of
Mississippi State University
in Partial Fulfillment of the Requirements
for the Degree of Master of Science
in the Department of Industrial Engineering

Mississippi State, Mississippi

August 1992

AN INTERACTIVE ARTIFICIAL CUTTING PLANE METHOD FOR BICRITERION INTEGER PROGRAMMING PROBLEMS

Accession For	
NTIS GRA&I	✓
DTIC TAB	☐
Unannounced	☐
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	

By

Diane Breivik Allen

Approved:

[illegible]
Wan Seon Shin
Associate Professor of Industrial Engineering (Major Professor)

Stanley F. Bullington
Stanley F. Bullington
Associate Professor and Graduate Coordinator of the Department of Industrial Engineering

[illegible]
Chuen-Lung Chen
Assistant Professor of Industrial Engineering

[illegible]
Larry G. Brown
Professor and Head of the Department of Industrial Engineering

[illegible]
Robert A. Altenkirch
Dean of the College of Engineering

Richard D. Koshel
Richard D. Koshel
Dean of the Graduate School

Name: Diane Breivik Allen

Date of Degree: August 7, 1992

Institution: Mississippi State University

Major Field: Industrial Engineering

Major Professor: Dr. Wan S. Shin

Title of Study: AN INTERACTIVE ARTIFICIAL CUTTING PLANE METHOD FOR BICRITERION INTEGER PROGRAMMING PROBLEMS

Pages in Study: 80

Candidate for Degree of Master of Science

This thesis develops an interactive solution method for bicriterion integer mathematical programming problems, called the Artificial Cutting Plane (ACP) method. This method consists of four major steps:

(1) Determine initial boundaries and the initial incumbent solution.

(2) Locate an associated frontier nondominated solution (AFNS) to the incumbent solution. If there is no AFNS in the remaining feasible area, the current incumbent solution is the best compromise solution and the algorithm terminates.

(3) Present the decision maker (DM) the incumbent solution and its AFNS, and update the boundaries based on which solution is preferred.

(4) Let the preferred solution be the new incumbent, and return to step 2.

The ACP method was tested and compared to Aksoy's Interactive Branch and Bound method on the basis of four evaluation criteria. Both methods were applied to randomly generated solutions based on five different shapes of efficient frontiers. Four different utility functions, with three variations each, were used to simulate the responses of a DM. The ACP method generated very competitive results against Aksoy's method.

ACKNOWLEDGEMENTS

Sincere appreciation is extended to Dr. Wan S. Shin for serving as my major advisor during my graduate studies and for providing substantial guidance and support during the course of preparing this thesis. I would like to thank Dr. C.-L. Chen and Dr. S. F. Bullington for serving on my committee. Appreciation is also extended to Dr. L. G. Brown and the Industrial Engineering Department for their general assistance during my graduate studies.

Financial support for this study and my graduate degree was provided by the U. S. Air Force. I am grateful for the outstanding opportunity I was granted.

Finally, deep appreciation and gratitude is extended to my husband, Aaron, without whose encouragement and infinite support this endeavor would not have been possible.

D. B. A.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

A common problem in applications of decision theory and mathematical programming deals with conflicting objectives. Especially common are problems concerning two criteria or objectives, such as finding a balance between cost and quality or compromising between conflicting interests of two decision makers.

Although much research has been done in the general area of multi-criteria decision making, relatively little work has exploited the simpler, specific case of two criteria. Most solution methods developed for multi-criteria decision making are applicable to solving bicriterion problems. It is possible, however, that one can develop an efficient solution method for bicriterion problems by making use of the relative simplicity of their problem structures. Also, when the idea behind a proposed method is unique, a preliminary study based on the bicriterion case helps test its applicability for general cases. The idea of this proposed method is unique.

This thesis develops an interactive algorithm for bicriterion integer mathematical programming problems, called the Artificial Cutting Plane (ACP) method. This algorithm iteratively introduces a constraint to reduce the decision space, depending on the decision maker's preference between two solutions. The efficiency of the algorithm is tested by solving randomly generated problems based on varying shapes of efficient frontiers. Finally, this thesis compares the new method with an existing solution method, Aksoy's Interactive Branch-and-Bound Algorithm (1990).

This thesis consists of five chapters and three appendices. In Chapter II, the problem statement and assumptions are explicitly stated, and a literature review of bicriterion mathematical programming is presented. The third chapter explains the proposed method. Specifically, this chapter explains the underlying theory associated with each step of the algorithm, including finding initial boundaries, locating an associated frontier nondominated solution (AFNS), interacting with the decision maker, and reducing the feasible region. The algorithmic procedure is formulated and a numerical example of the algorithm is given. The fourth chapter describes a comparative study of the ACP method and Aksoy's branch and bound method. Conclusions are presented in the fifth chapter. The appendices contain listings of the three computer programs used in the comparative study.

CHAPTER II

PROBLEM STATEMENT AND LITERATURE REVIEW

2.1 Problem Statement and Assumptions

The bicriterion integer mathematical programming problem (BIMP) under consideration is of the following form:

(BIMP) Maximize $f_1(x)$

Maximize $f_2(x)$

subject to: $x \in X;\ x \in I$ (Integer)

$f_1(x)$ and $f_2(x)$ are two concave, conflicting objective functions. x is an n-dimensional vector of decision variables. X is the decision space and it is assumed to be convex and compact. A bicriterion problem can also exist in pure decision theory problems where known values of two conflicting attributes replace the optimization functions.

In practical applications, it is rare for a single solution to maximize both functions. Therefore, the problem becomes finding the best compromise solution which optimizes the decision maker's implicitly known preference or utility function. The utility function represents the decision maker's preferences between values of the two objective functions: $U[f_1(x),\ f_2(x)]$ such that $U[f_1(x^1),\ f_2(x^1)] > U[f_1(x^2),\ f_2(x^2)]$ if and only if the decision maker prefers $[f_1(x^1),\ f_2(x^1)]$ to $[f_1(x^2),\ f_2(x^2)]$. This utility function is assumed to be increasing and quasiconcave. The requirements for quasiconcavity of the utility function and convexity of the decision space ensure that the best compromise solution is a global optimum.

2.2 Literature Review

As mentioned earlier, extensive research has been done in the general area of multi-criteria decision making. For a survey of literature on interactive methods applied to continuous variables, see Shin and Ravindran (1991). Other reviews of general multi-criteria problems and solution methods are available in Cochrane and Zeleny (1973), Eswaran (1983), Evans (1984), Steuer (1986), and references therein.

Early work in the area of bicriterion problems was done by Geoffrion in 1967 and Pasternak and Passy in 1973. Geoffrion's algorithm uses parametric programming and is applied to continuous cases, while Pasternak and Passy's method is applied to Boolean variables. However, in these works the authors assume that the utility function, which specifies the decision maker's preference between the two objectives, is explicitly known. In actual situations, this is usually not the case and it can be very difficult to develop even a reasonable estimate of the utility function. These two methods also concentrate on generating all efficient solutions rather than finding the best compromise solution. Benson (1979) presented a necessary and sufficient condition for a point to be an efficient point when both objective functions are concave over a convex set. He used this result in a parametric procedure for generating all efficient solutions without unnecessarily generating nonefficient solutions. Cohon, Church and Steer (1979) developed an algorithm to approximate the set of efficient solutions in a bicriterion problem, which lets the user control maximum possible error. Chalmet, LeMonidis and Elzinga (1986) developed an algorithm which locates all nondominated (efficient) points for a bicriterion integer programming problem, with special emphasis on the linear 0-1 case.

In 1978, Walker developed an interactive method of solving bicriterion problems which obtains a best compromise solution. This method presents the decision maker (DM) with the two objective function values for a point in the decision space, and asks the DM yes or no questions regarding his or her satisfaction and whether or not one objective can be increased at the expense of the other. This requires the DM to know specific trade-off values, which can be a cognitive burden. Sadagopan and Ravindran (1982) developed two interactive algorithms to locate the best compromise solution to a continuous bicriterion mathematical programming problem. Their methodology is based on a constrained criteria approach, meaning that one objective function is systematically used as a constraint while the other is maximized. The Paired Comparison Method

(PCM) presents the DM with two efficient solutions to which the DM must respond which is preferred, if either, and the objective space is reduced depending on the response. Their Comparative Tradeoff Method (CTM) also uses a region elimination scheme, but presents the DM with a tradeoff value and asks if he or she would be willing to trade more, less, or an equal amount of one criterion for a unit increase in the other. Both of these methods significantly reduce the burden to the DM as opposed to requiring precise local tradeoffs or presenting all efficient solutions to the DM for selection. Eswaran, Ravindran and Moskowitz (1989) developed a non-interactive method as well as an interactive, region elimination procedure for integer problems which utilizes paired comparisons and Tchebycheff norms. Unfortunately, this method requires a number of difficult assumptions such as a uniformly dominant efficient set, supported efficient points, and unimodality. In 1990, Aksoy developed a branch-and-bound algorithm for mixed integer, bicriterion problems which also utilizes paired comparisons. This method has very broad applications since it can be applied to continuous, integer and mixed cases, and it does not require convexity of the decision space or concavity of the utility function. It does, however, ask the DM to compare infeasible ideal solutions.

In general, interactive approaches to bicriterion mathmatical programming problems have a number of advantages. The DM is not required to consider all of the efficient solutions. By successively providing her preference about the current solution either implicitly or explicitly, the DM progressively gains knowledge about her preference structure. Also, a best compromise solution obtained using this approach has a better chance of being accepted and implemented, since the DM is involved in the solution process.

Other research has been done on bicriterion problems in the areas of linear programming (Adulbhan and Tabucanon 1977; Kiziltan and Yucaoglu 1981), scheduling (John and Sadowski 1986), transportation (Aneja and Nair 1979; Malhotra 1982), and shortest path problems (Climaco and Martins 1981; Henig 1985). Although the number of studies on bicriterion problems has increased in recent years, there are still relatively few algorithms in this area which use an interactive approach or which are applicable to integer problems.

CHAPTER III

AN INTERACTIVE ARTIFICIAL CUTTING PLANE METHOD

The purpose of solution methods of BIMPs is to find the best compromise solution. Steuer (1986, p. 148) and others have shown that the best compromise solution, which maximizes the decision maker's utility function, must be a nondominated solution.

DEFINITION 1: Nondominated Solution

A solution $[f_1(x^*),\ f_2(x^*)]$ is nondominated if there exists no other point such that $f_1(x) \geq f_1(x^*)$ and $f_2(x) \geq f_1(x^*)$, with at least one inequality strictly greater than. In such a case, the point x^* is called an efficient point.

The interactive artificial cutting plane method is composed of four basic steps. In Step 1, initial boundaries are determined by separately maximizing each objective function. One of these solutions becomes the initial incumbent solution. Step 2 consists of locating an associated frontier nondominated solution (AFNS) of the incumbent solution. In Step 3, the decision maker (DM) is presented with the incumbent solution and its AFNS and asked which is preferred. Depending on the DM's response, a constraint is added to reduce the decision space in Step 4. These steps are repeated until only one solution remains which is the best compromise solution.

3.1 Finding Initial Boundaries

Step 1 of the algorithm locates the initial boundaries of the best compromise solution by separately maximizing each objective function; that is, solving the nonlinear programming problems (Aksoy 1990, 406):

(1) Maximize $f_i(x)$, subject to $x \in X$ and $x \in I$; and

(2) Maximize $f_j(x)$, subject to $x \in X^*$

where $i = 1, 2$ and X^* is the set of solutions for problem (1). When $i = 1$ and $j = 2$, let the solution to problem (2) be $r = (r_1, r_2) = [f_1(x^r), f_2(x^r)]$. When $i = 2$ and $j = 1$, let the solution to problem (2) be $l = (l_1, l_2)$. If these problems both result in the same solution, this is the best compromise solution. Otherwise, set the initial boundaries to $f_1(x) \geq l_1$ and $f_2(x) \geq r_2$. By definition, we also know that $f_1(x) \leq r_1$ and $f_2(x) \leq l_2$, although these constraints do not need to be explicitly stated since they do not eliminate any solutions in the region.

LEMMA 1. The initial boundaries include the best compromise solution.

PROOF. First, assume there exists a solution $f^* = [f_1(x^*), f_2(x^*)]$ such that $f_1(x^*) > r_1$ or $f_2(x^*) > l_2$. Then, by definition of r and l, $f_1(x^*) > \text{Max}\ \{f_1(x)\}$ or $f_2(x^*) > \text{Max}\ \{f_2(x)\}$, which cannot be true. Therefore, $f_1(x) \leq r_1$ and $f_2(x) \leq l_2$. Next, suppose there exists a solution f^b such that $f_1(x^b) < l_1$. From above, we know that $f_2(x^b) < l_2$. By definition, f^b is dominated by l and thus cannot be the best compromise solution. A similar argument shows that the constraint $f_2(x) \geq r_2$ does not eliminate the best compromise solution. QED.

3.2 Locating an Associated Frontier Nondominated Solution (AFNS)

After setting the initial boundaries, solution l will serve as the initial incumbent solution for comparison. The DM will compare this solution to one of its associated frontier nondominated solutions.

DEFINITION 2: Extreme Nondominated Solution

An extreme nondominated solution is a nondominated solution which cannot be represented as (or dominated by) a convex combination of any two other feasible solutions.

DEFINITION 3: Adjacent Extreme Nondominated Solution (AENS)

Let $f^1 = [f_1(x^1), f_2(x^1)]$ and $f^2 = [f_1(x^2), f_2(x^2)]$ be two extreme nondominated solutions to a bicriterion problem, and x^1 and x^2 be the respective extreme efficient points. Assume without loss of generality that $f_1(x)$ increases and $f_2(x)$ decreases as the solution moves from x^1 to x^2. Then f^1 and f^2 are adjacent extreme nondominated solutions and x^1 and x^2 are adjacent extreme efficient points if there exists no extreme point $x \in X$ such that $f_1(x^1) < f_1(x) < f_1(x^2)$ and $f_2(x^1) > f_2(x) > f_2(x^2)$.

DEFINITION 4: Associated Frontier Nondominated Solution (AFNS)

Let f^1 and f^2 be nondominated solutions. Let set F =

$$\{(f_1(x),\ f_2(x)) \mid \frac{[f_2(x^2) - f_2(x^1)]}{[f_1(x^2) - f_1(x^1)]} [f_1(x)-f_1(x^1)] + [f_2(x^1)-f_2(x)] > 0\}.$$

f^1 and f^2 are associated frontier nondominated solutions if F = $\varnothing$ (empty set). In other words, if a half-space extending from a line connecting two or more nondominated solutions contains no other solutions, the points on that line are AFNS's. Note that AENS's are a subset of AFNS's and the corresponding AEEPs are a subset of associated frontier efficent points (AFEPs).

In this algorithm, AFNS's are determined in the following manner. Graphically, starting from the "leftmost point" (the nondominated solution, 1, with max $f_2(x)$), a horizontal line extending from this point is swung downward to the right until it intersects a point, which is an AFNS (see figure 3.1). This equates to locating the point which forms a line with the least negative slope when connected to point 1. If there is more than one point on this line, the point farthest from the originating point is its AENS, since a solution between them could be represented as a convex combination of the outer two points and thus is not extreme. However, any point on this line is an AFNS and could be selected for comparison without altering the convergence of the algorithm.

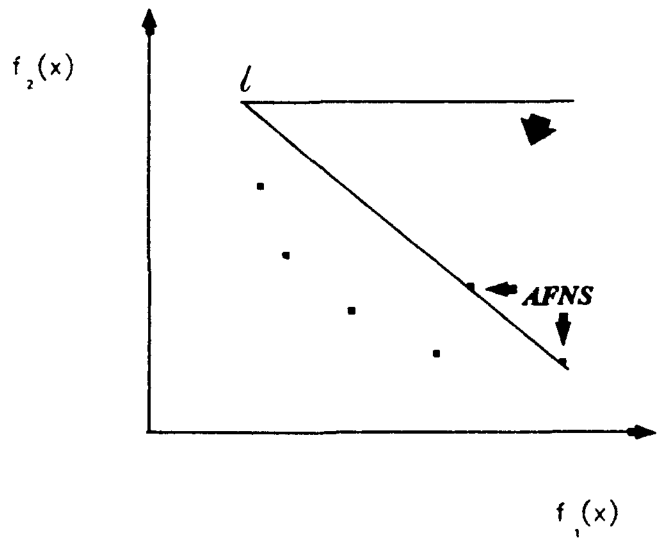

Figure 3.1 Finding AFNS to the right of point 1.

Mathematically, AFNSs to the right of 1 can be found by solving the following single objective optimization problem:

(P_b-1) Maximize b

subject to:

$$[f_1(x) - f_1(x^l)] + b[f_2(x) - f_2(x^l)] \geq 0;$$

$$f_1(x) > f_1(x^l) \text{ and } b > 0;$$

$f_2(x) > c$, where c is the current boundary constraint on $f_2(x)$.

Note that the second constraint ensures that only points to the right of the originating point are checked. Since the algorithm only uses originating points which are efficient, the constraint $b > 0$ is actually redundant.

When all values of $f_1(x)$ and $f_2(x)$ are known, as will be the case when comparing two attributes as opposed to solving a nonlinear programming problem, the above problem reduces to:

$$\text{Maximize } b = \frac{f_1(x) - f_1(x^l)}{f_2(x^l) - f_2(x)}; \quad f_1(x) > f_1(x^l) \text{ and } f_2(x) > c$$

This modified format will also be used for the example and comparative study of the algorithm.

To find an AFNS when "swinging left" from a point r, the following P_b problem must be solved:

(P_b-2) Minimize b

subject to:

$$[f_1(x) - f_1(x^r)] + b[f_2(x) - f_2(x^r)] \geq 0$$

$$f_2(x) \geq f_2(x^r)$$

$$f_1(x) \geq c.$$

Note that these two P_b problems are equivalent, except that they use the opposite point as the originating point (see figure 3.2).

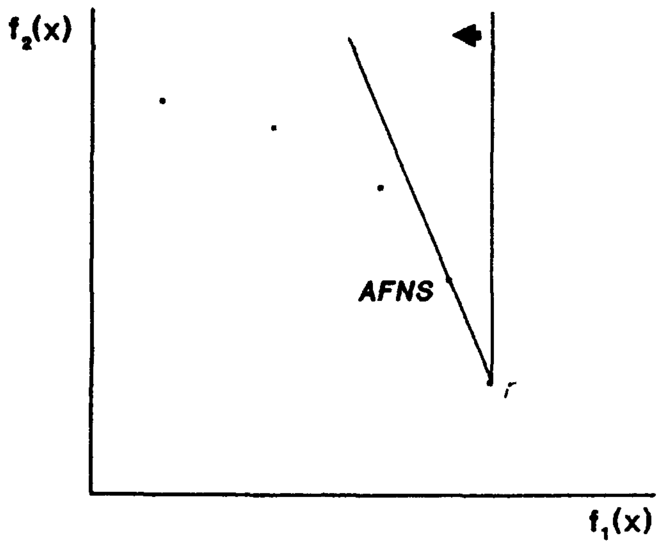

Figure 3.2 Finding AFNS to the left of point r using P_b-2.

THEOREM 1. Given an initial extreme nondominated solution (ENS), if there exists an optimal solution to problem P_b-1, it is an AFNS.

PROOF. Let l be the initial ENS and let b^* be the optimal solution to P_b-1, located at $f^* = [f_1(x^*),\ f_2(x^*)]$.

From P_b-1: $(f_1(x) - f_1(x^l)) + b(f_2(x) - f_2(x^l)) \geq 0$

$$b \leq \frac{(f_1(x^l) - f_1(x))}{(f_2(x) - f_2(x^l))}.$$

$[(f_2 - f_2^l) < 0$ by other constraints$]$

Since b^* = maximum b,

$$\frac{(f_1(x^l) - f_1(x^i))}{(f_2(x^i) - f_2(x^l))} \leq b^* = \frac{(f_1(x^l) - f_1(x^*))}{(f_2(x^*) - f_2(x^l))}$$

for all other solutions $[f_1(x^i),\ f_2(x^i)]$. Suppose f^* is not an AFNS to point l. Then by definition of AFNS, there exists at least one solution $[f_1(x),\ f_2(x)]$ in F such that

$$\frac{[f_2(x^*) - f_2(x^l)]}{[f_1(x^*) - f_1(x^l)]}\,[f_1(x)-f_1(x^l)] + [f_2(x^l)-f_2(x)] > 0.$$

By rearranging and applying the other constraints of the P_b problem, this can be rewritten

$$\frac{[f_1(x^l) - f_1(x^i)]}{[f_2(x^i) - f_2(x^l)]} > \frac{[f_1(x^l) - f_1(x^*)]}{[f_2(x^*) - f_2(x^l)]}$$

which contradicts the assumption that f^* is the optimal solution to P_b-1. Therefore, set F must be empty and solutions l and f^* are AFNS's. QED.

COROLLARY 1: Given an initial ENS, if there exists an optimal solution to P_b-2 problem, it is an AFNS.

PROOF: Similar to proof of Theorem 1. QED.

3.3 Interacting with the Decision Maker (DM)

The algorithm proceeds by asking the DM to compare two AFNS's, l and r, which were located using the above P_b problems. The DM will indicate whether solution l is preferred to r, r is preferred to l, or she is indifferent between solutions l and r. Based on his or her response, a constraint is constructed in order to reduce the solution space. Suppose that l and r are two AFNS's such that $f_1(x^r) > f_1(x^l)$ and $f_2(x^l) > f_2(x^r)$. If the DM prefers l to r, a constraint is added such that $f_2(x) > f_2(x^r)$. If the DM prefers r to l, the constraint $f_1(x) > f_1(x^l)$ is added. This elimination procedure should not eliminate the best compromise solution. It will be shown that any solution in the eliminated region must either be a dominated solution or contained in a dominated convex cone, which is constructed using the results of the pairwise comparison. The convex cone, defined below, is constructed using solutions l and r, and the ideal solution which dominates both l and r. Since, by definition, the ideal solution will be preferred to both l and r, whichever solution was not preferred by the DM in the pairwise comparison will be the least preferred solution of the three.

THEOREM 2. Assume a quasiconcave function defined in a 2-dimensional Euclidean space, $U(f_1(x),f_2(x))$. Consider three distinct solutions, $f(x^i) = (f_1(x^i), f_2(x^i))$, and assume that $U(f^k) \leq U(f^i)$, $k \neq i$. If $z \in Z$ and $z \neq f^k$, where $Z = \{ z | z = f^k + \Sigma_{i \neq k} \mu_i(f^k - f^i), \mu_i \geq 0 \}$, it follows that $U(z) \leq U(f^k)$. See figure 3.3.

PROOF. See Korhonen, Wallenius and Zionts, 1984.

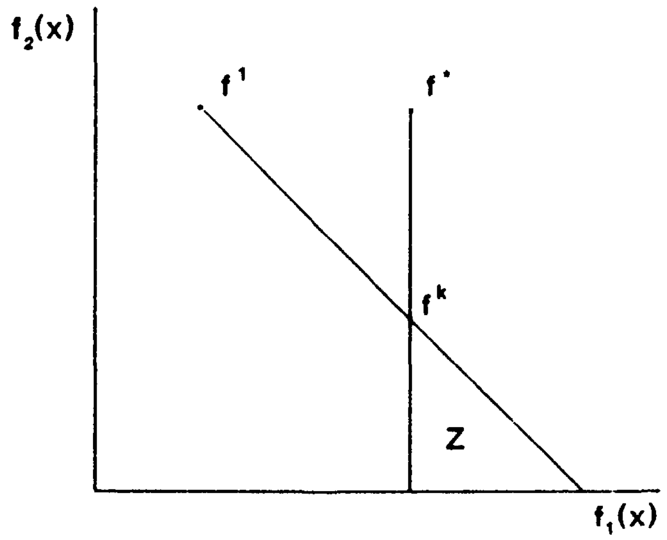

Figure 3.3 Illustration of Lemma 2.

3.4 Reducing the Feasible Region

For the theory of eliminating non-optimal areas for this algorithm, we shall assume that f^k as defined above is the DM's least preferred solution from a pairwise comparison. The two other solutions used to construct the dominated convex cone are f^1, the DM's preferred solution from the same pairwise comparison, and f^*, the ideal (infeasible) solution which dominates both other solutions.

THEOREM 3. Consider two AFNS's, $f^1 = [f_1(x^1),\ f_2(x^1)]$ and $f^2 = [f_1(x^2),\ f_2(x^2)]$, where $f_1(x^2) > f_1(x^1)$ and $f_2(x^1) > f_2(x^2)$. Assume that f^1 is preferred to f^2; that is, $U(f^1) > U(f^2)$. The best compromise solution, Max U, does not lie in the region such that $f_2(x) \leq f_2(x^2)$.

PROOF. Partition the region $f_2(x) \leq f_2(x^2)$ into the following subsets (see figure 3.4):

$$A = \{(f_1,\ f_2) \mid\ f_2 \leq f_2^2 \text{ and } f_1 \leq f_1^2\}$$

$$B = \{(f_1,\ f_2) \mid\ \frac{(f_2^2 - f_2^1)}{(f_1^2 - f_1^1)} (f_1 - f_1^1) + (f_2^1 - f_2) \leq 0 \text{ and } f_1 > f_1^2\ \}$$

$$C = \{(f_1,\ f_2) \mid\ \frac{(f_2^2 - f_2^1)}{(f_1^2 - f_1^1)} (f_1 - f_1^1) + (f_2^1 - f_2) > 0 \text{ and } f_2 \leq f_2^2\ \}$$

(Where f_1^1 is shorthand notation for $f_1(x^1)$.)

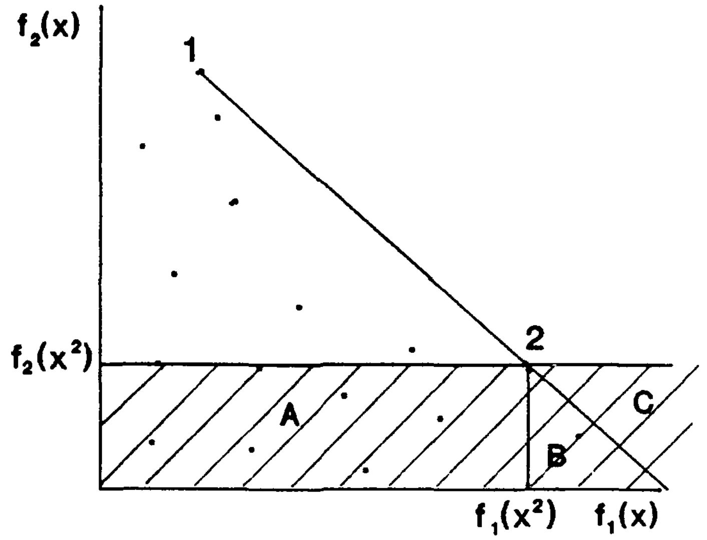

Figure 3.4 Illustration of eliminated areas when 1 is preferred to 2. Area A is dominated by 2; B is contained in non-optimal convex cone; C is empty.

By definition, any solution in subset A is dominated by f^2 and thus is not the best compromise solution. Next, consider a solution in subset B. Since $U(f^1) > U(f^2)$, we know that f^2 is the least preferred solution of f^1, f^2 and f^*, where $f^* = [f_1(x^2), f_2(x^1)]$, the ideal solution which dominates both other solutions. The area B is equivalent to a convex cone created using these three solutions, and thus does not contain the best compromise solution according to Theorem 2. Finally, set C is a subset of set F from the definition of AFNS. Since F is an empty set, subset C is also an empty set and thus cannot contain the best compromise solution. QED.

COROLLARY 2: Consider solutions $f^1 = [f_1(x^1), f_2(x^1)]$ and $f^2 = [f_1(x^2), f_2(x^2)]$, where $f_1(x^2) > f_1(x^1)$ and $f_2(x^1) > f_2(x^2)$. Assume that f^2 is preferred to f^1; that is, $U(f^2) > U(f^1)$. The best compromise solution does not lie in the region such that $f_1(x) \leq f_1(x^1)$.

PROOF: Similar to proof of Theorem 3. See figure 3.5. QED.

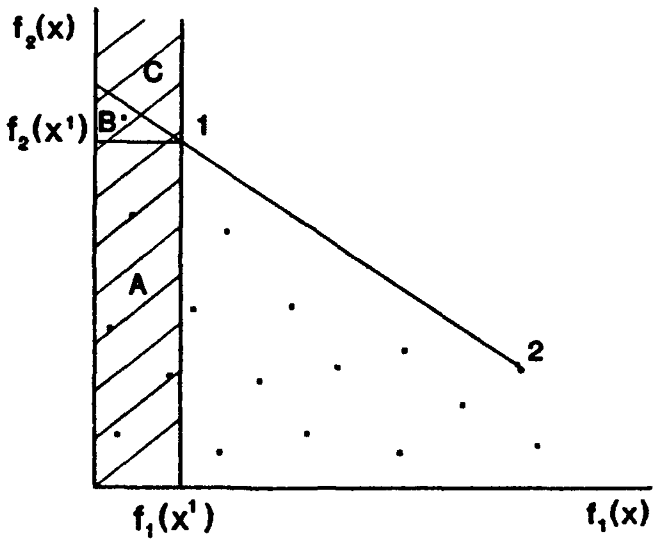

Figure 3.5 Eliminated areas when 2 is preferred to 1. Area A is dominated by 1; B is non-optimal convex cone; C is empty.

LEMMA 2. The addition of constraints such as $f_1(x) > c$ and $f_2(x) > c$, c a constant, does not alter the convexity of the decision space, X.

PROOF. It is already assumed that the initial decision space is convex and that $f_1(x)$ and $f_2(x)$ are concave functions. This implies that the sets

$\{x \mid f_i(x) - c > 0;\ i = 1,2\}$ are convex sets. Since the intersection of two convex sets is also a convex set, adding such constraints maintains the convexity of X. QED.

We are now ready to explicitly state the steps of this algorithm.

3.5 The Algorithmic Procedure

STEP 1: Determine the initial boundaries by solving the following nonlinear programming problems:

(1) Maximize $f_i(x)$, subject to $x \in X$ and $x \in I$; and

(2) Maximize $f_j(x)$, subject to $x \in X^*$

where $i, j \in \{1, 2\}$, $i \neq j$, and X^* is the set of solutions for problem (1). When $i = 1$ and $j = 2$, let the solution to problem (2) be $r = (r_1, r_2) = [f_1(x^r), f_2(x^r)]$. When $i = 2$ and $j = 1$ let the solution to problem (2) be $l = (l_1, l_2)$. If both problems result in the same solution, this is the best compromise solution. STOP. Otherwise, set the initial boundaries to $f_1(x) \geq l_1$ and $f_2(x) \geq r_2$. Let l be the initial solution for comparison. Using this notation, the solution l will always represent the graphical "lefthand" comparison solution, and r will represent the "righthand" comparison solution.

STEP 2: Find an AFNS to the right of l by solving the first P_b problem:

(P_b-1) Maximize b

subject to:

$[f_1(x) - f_1(x^l)] + b[f_2(x) - f_2(x^l)] \geq 0$

$f_1(x) > f_1(x^l)$

$f_2(x) > c$, where c is the current boundary constraint on $f_2(x)$.

If there is a solution to this problem, let this solution be solution r, the second solution for comparison.

If there is no solution to the first P_b problem, then there are no solutions to the right of l. Set solution r equal to solution l; it will now be considered the righthand solution. Swing left from r by solving the second P_b problem:

(P_b-2) Minimize b

subject to:

$[f_1(x) - f_1(x^r)] + b[f_2(x) - f_2(x^r)] \geq 0$

$f_2(x) \geq f_2(x^r)$

$f_1(x) \geq c$.

If there is no solution to this P_b problem, STOP. Solution r is the best compromise solution.

STEP 3: Ask the Decision Maker to compare solutions l and r. If solution l is preferred to r, add the constraint $f_2(x) > r_2$. If solution r is preferred to l, add the constraint $f_1(x) > l_1$. If the DM is indifferent between the two solutions, add the constraints $f_2(x) > r_2$ and $f_1(x) \geq l_1$.

STEP 4: Let the preferred solution be solution 1, the first solution for the next comparison. Go to Step 2.

Since an integer programming problem has a finite number of solutions, and this algorithm eliminates at least one solution at each iteration without generating additional solutions, it will find the best compromise solution in a finite number of iterations.

3.6 Numerical Example

The following is a simple example which shows the stepwise procedure of the above algorithm. Assume that the DM's implicitly known, concave utility function is:

Maximize $U(f_1(x), f_2(x)) = -[f_1(x) - 5]^2 - [f_2(x) - 6]^2$

This will be used only to determine the DM's pairwise preferences.

The following solutions will be used for this example problem. These solutions could represent attributes, as found in decision theory, or solutions to nonlinear programming problems. In actual applications of this algorithm, nonlinear programming problems would be solved to locate these solutions.

Solutions	$f_1(x)$	$f_2(x)$	$U(f_1(x),f_2(x))$
A	1	6	-16
B	0.5	4	-24.25
C	1.5	4.5	-14.5
D	1	2.5	-28.25
E	2	3.5	-15.25
F	2	1	-34
G	3	2	-20
H	4	0.5	-31.25
I	5	1	-25
J	3.1	1.1	-27.62

Note that solutions B, D, F, and H are dominated solutions.

STEP 1: Determine initial boundaries. Max $f_1(x) = 5$, corresponding to solution I = (5,1). Max $f_2(x) = 6$, corresponding to solution A = (1,6). Initial constraints become $f_1(x) \geq 1$ and $f_2(x) \geq 1$. This eliminates solutions B and H. Set A = 1, the initial comparison solution (see figure 3.6).

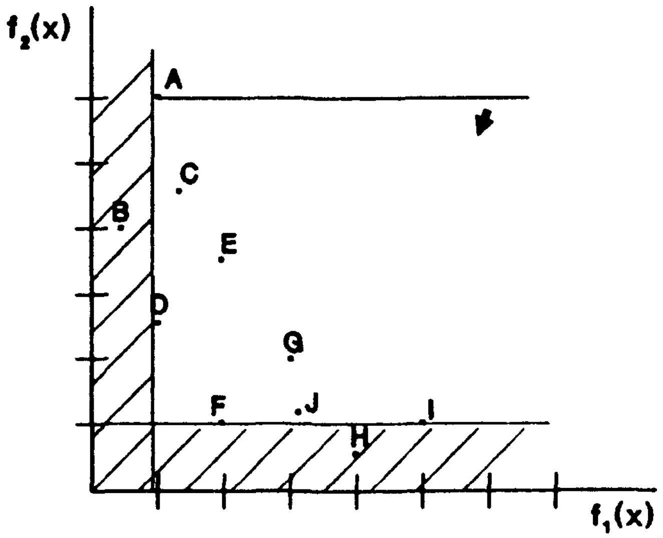

Figure 3.6 Initial boundaries for example problem.

Iteration 1

STEP 2: "Swing right" from solution A by solving the P_b problem:

$$\text{Maximize } b = \frac{f_1(x) - 1}{6 - f_2(x)}$$

subject to $f_1(x) > 1$ and $f_2(x) \geq 1$.

The solution to this problem is b = 0.8, found at solution I = (5,1). Let I be solution r, the next solution for comparison.

STEP 3: Ask the DM to compare solutions A and I. Based on the implicit utility function, the DM will prefer A to I. Replace the initial constraint $f_2(x) \geq 1$ with the more restrictive constraint $f_2(x) > 1$. This eliminates I and F, which is dominated by I.

STEP 4: A, the preferred solution, remains the incumbent solution l for the next comparison. Go to Step 2.

Iteration 2

STEP 2: "Swing right" from solution A again, using the same P_b problem as above but with the new constraints. The solution is b = 0.5, which corresponds to solution G, the new AFNS to A. Let G = r, the second solution for comparison.

STEP 3: Ask the DM to compare A and G; A is preferred to G. Add the constraint $f_2(x) > 2$, which makes the constraint $f_2 > 1$ redundant. The new constraint eliminates solutions G and J.

STEP 4: Again, A remains designated as 1 for the next comparison. Go to Step 2.

Iteration 3

STEP 2: "Swing right" from solution A again, using the same P_b problem as above but with the new constraint. The solution is $b = 0.4$, which corresponds to solution E, the new AFNS to A. Let $E = r$, the second solution for comparison.

STEP 3: Ask the DM to compare A and E; E is preferred to A. The original constraint $f_1(x) \geq 1$ is replaced by the more restrictive constraint $f_1(x) > 1$. This will eliminate solution A and solution D, which is dominated by A.

STEP 4: The new preferred solution, E, becomes 1, the new incumbent solution for comparison. Go to Step 2.

Iteration 4

STEP 2: "Swing right" from E by solving the P_b problem:

$$\text{Maximize } b = \frac{f_1(x) - 2}{3.5 - f_2(x)} ;$$

subject to: $f_1(x) > 2$; (only checks solutions to the right of E)

$f_2(x) > 2$.

This problem has no solution, meaning that no solutions remain to the right of E. Let $E = r$ and "swing left" from E by solving the second P_b problem:

$$\text{Maximize } b = \frac{3.5 - f_2(x)}{f_1(x) - 2};$$

subject to: $f_1(x) < 2$; (only checks solutions to the left of E)

$f_1(x) > 1$;

$f_2(x) > 2$;

$b > 0$.

The solution to this problem is $b = 2$, corresponding to solution C, the only feasible solution. Let C = 1, the second solution for comparison.

STEP 3: Present the DM with the pairwise comparison of E and C; according to the implicit utility function C is preferred to E. Add the constraint $f_2(x) > 3.5$, which eliminates both E and F, which is dominated by E.

STEP 4: The most preferred solution, C, becomes 1 for the next comparison. Go to Step 2.

Iteration 5

STEP 2: Swing both right and left from C by sequentially solving the two P_b problems. Neither of these problems will have a solution, meaning that C is the best compromise solution. STOP.

CHAPTER IV

THE ARTIFICIAL CUTTING PLANE METHOD AND AKSOY'S METHOD: A COMPARATIVE STUDY

In this chapter, the Artificial Cutting Plane (ACP) method is compared with Aksoy's (1990) Branch and Bound method applied to pure integer, bicriterion programming problems. Both methods were programmed in Turbo Pascal and run on a personal computer. The comparative study applies each method to randomly generated solutions based on a specified efficient frontier shape rather than solving actual mathematical programming problems at each iteration. This is done for testing purposes only, as illustrated in the numerical example of Chapter III. However, this method could also be used when solving a decision theory problem with two known attributes.

To simulate possible preferences of a real DM, four different types of utility functions are used - linear, quadratic, fourth power and exponential. Three variations of each utility function are used. These utility functions are applied to solution spaces representing five different shapes of efficient frontiers. Each general shape was generated using five different variances.

4.1 Utility Functions Tested

Because there is no actual DM, this comparison uses four different utility functions to simulate possible preferences. These utility functions are used to select between two solutions presented to the function, representing the DM. The programs will select the solution with the highest utility function value as the most preferred solution for that iteration. The functions used are linear, quadratic, fourth power and exponential, with three variations for each utility function. The actual test utility functions are as follows:

Linear:

(1) $5 * f_1(x) + 5 * f_2(x)$

(2) $8 * f_1(x) + 2 * f_2(x)$

(3) $2 * f_1(x) + 8 * f_2(x)$

Quadratic:

(1) $-5[f_1(x^*) - f_1(x)]^2 - 5[f_2(x^*) - f_2(x)]^2$

(2) $-8[f_1(x^*) - f_1(x)]^2 - 2[f_2(x^*) - f_2(x)]^2$

(3) $-2[f_1(x^*) - f_1(x)]^2 - 8[f_2(x^*) - f_2(x)]^2$

Fourth Power:

(1) $-5[f_1(x^*) - f_1(x)]^4 - 5[f_2(x^*) - f_2(x)]^4$

(2) $-8[f_1(x^*) - f_1(x)]^4 - 2[f_2(x^*) - f_2(x)]^4$

(3) $-2[f_1(x^*) - f_1(x)]^4 - 8[f_2(x^*) - f_2(x)]^4$

Exponential:

(1) $5[1 - e^{-0.08f1(x)/f1(x^*)}] + 5[1 - e^{-0.08f2(x)/f2(x^*)}]$
$+10[(1 - e^{-0.08f1(x)/f1(x^*)})(1 - e^{-0.08f2(x)/f2(x^*)})]$

(2) $8[1 - e^{-0.12f1(x)/f1(x^*)}] + 2[1 - e^{-0.04f2(x)/f2(x^*)}]$
$+10[(1 - e^{-0.12f1(x)/f1(x^*)})(1 - e^{-0.04f2(x)/f2(x^*)})]$

(3) $2[1 - e^{-0.04f1(x)/f1(x^*)}] + 8[1 - e^{-0.12f2(x)/f2(x^*)}]$
$+10[(1 - e^{-0.04f1(x)/f1(x^*)})(1 - e^{-0.12f2(x)/f2(x^*)})]$

The linear, quadratic and fourth power functions are all quasiconcave, in accordance with the assumptions of the ACP method. The exponential functions were not checked for quasiconcavity. They are adapted from Ramachandran (1989).

4.2 Shapes of Efficient Frontiers Tested

The ACP method and Aksoy's method, using the above utility functions, were tested on 25 different sets of randomly generated solutions. These 25 sets were based on five different shapes of efficient frontiers, each with variances of 1, 3, 5, 7 and 9. The efficient frontier shapes used were linear, concave, convex, S-shaped and reverse S-shaped. They are illustrated in figures 4.1 through 4.5.

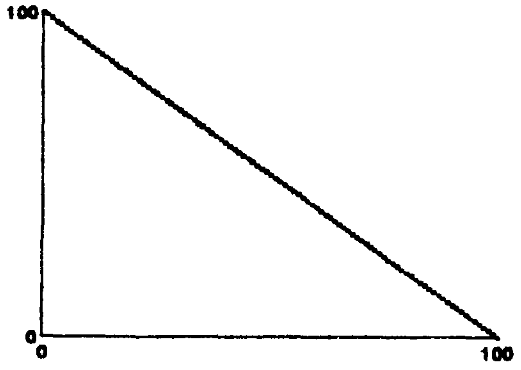

Figure 4.1 Linear Efficient Frontier

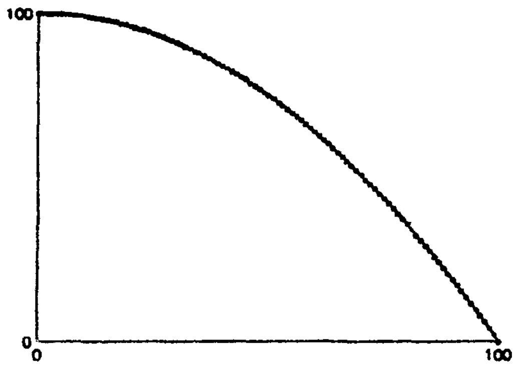

Figure 4.2 Concave Efficient Frontier

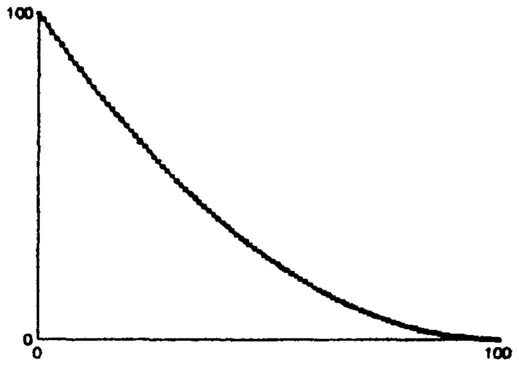

Figure 4.3 Convex Efficient Frontier

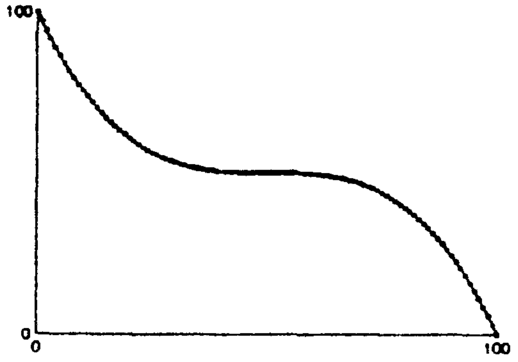

Figure 4.4 S-Shaped Efficient Frontier

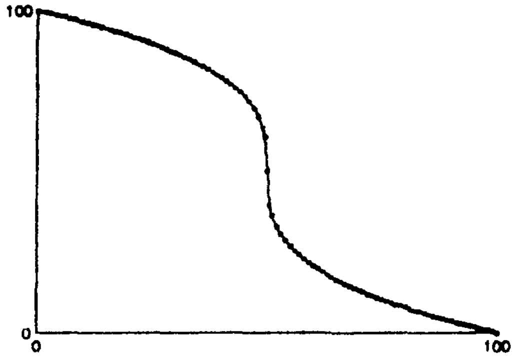

Figure 4.5 Reverse S-Shaped Efficient Frontier

The data sets were all generated using a program written in Turbo Pascal. First, 500 numbers were randomly generated between 0 and 100, representing the values of $f_1(x)$. A specified function was used to generate the appropriate shape of the efficient frontier, and each value of $f_1(x)$ was passed to that function to determine the mean of $f_2(x)$. Each value representing $f_2(x)$ was then randomly generated using the polar Box-Muller method (Dagpunar 1988, 93) based on a Normal distribution with a mean equal to the corresponding function value. Each frontier shape was generated five times, with variance ranging from 1 to 9. Therefore, each data set contains 500 bicriterion solutions. The number of efficient solutions in each set would tend to decrease as variability increased, since some solutions generated would have a greater probability of dominating other solutions, with each data set containing a maximum of 500 (100%) efficient solutions. The program used to generate the solution sets is included in Appendix A. The functions which were used to generate each shape are as follows:

(1) Linear: $f_2(x) = 100 - f_1(x)$

(2) Concave: $f_2(x) = 100 - 0.01[f_1(x)]^2$

(3) Convex: $f_2(x) = 0.01[f_1(x) - 100]^2$

(4) S-Shaped: $f_2(x) = -0.0004[f_1(x) - 50]^3 + 50$

(5) Reverse S-Shaped: $f_2(x) = -(50)^{2/3}[f_1(x) - 50]^{1/3} + 50$

4.3 Evaluation Criteria

In this comparative study, four different evaluation criteria are used to investigate the effectiveness of each method and to measure the cognitive burden to the DM. The evaluation criteria used are described below:

(1) **The number of interactions with the DM.** This represents the number of pairwise comparisons required of the DM to solve a bicriterion problem by each method. It is a very critical criterion used to measure the cognitive burden on the DM.

(2) **The number of mathematical programming problems solved.** In actual applications of both methods, at least one nonlinear, single objective, mathematical programming (MP) problem is solved at each iteration. This criterion is indicative of the computation time which would be required to solve an actual bicriterion problem.

(3) **The average utility function difference between the pair of solutions presented to the DM in each iteration.** This criterion also indicates, to a lesser extent, the cognitive burden placed on the DM. If the utility function difference between two solutions is larger, it will be easier for the DM to specify her preference.

(4) **The average utility function difference between the preferred solution at each iteration and the ideal**

solution. This criterion compares the utility function value of the DM's preferred solution from any pairwise comparison with the utility function value of the data set's ideal solution. It is useful if the DM decides to terminate the solution process before reaching a best compromise solution, since a utility function value closer to the utility function value of the ideal solution would be preferred.

4.4 Comparison Based on Evaluation Criteria

The ACP method and Aksoy's method were both programmed in Turbo Pascal and compared based upon the four evaluation criteria described in section 4.3. The programs used to evaluate the two methods are included in Appendices B and C. Each program was run using the twelve variations of utility functions desribed in section 4.1. Each utility function was applied to each of the 25 solution sets described in section 4.2. Therefore, each program was run a total of 300 times. As expected, both methods obtained the same best compromise solution in every case.

4.4.1 Number of Interactions with the DM

The two methods were compared on the basis of the number of pairwise comparisons required of the DM to find the best compromise solution. In each case, the DM (or for testing, the utility function) is presented with two solutions and asked which if preferred, if either. The ACP method always asks the DM to compare two feasible solutions. Aksoy's branch and bound method sometimes requires the DM to compare a feasible solution with an infeasible, ideal solution of a node. Regardless of whether or not the solutions are feasible, this is a very important criteria for interactive methods, because it represents the cognitive burden required of the DM. Also, as the number of interactions increases, so does the possibility that the DM responds inconsistently.

When averaged over all 300 runs of each method, the ACP method required a average of 15.83 paired comparisons before reaching the best compromise solution, with a standard deviation of 13.39. Aksoy's branch and bound method required an average of 26.18 paired comparisons with a standard deviation of 18.29. The breakdown of this criterion for the four different utility functions and five general shapes of the efficient frontier is presented in table 4.1. In every case, the ACP method requires the same or fewer interactions with the DM than Aksoy's method. The ACP method performs better with a linear or exponential utility function than with a quadratic or fourth power utility function. It also requires fewer comparisons, on the average, when the efficient frontier is generally linear or concave, and requires the most comparisons when the efficient frontier is convex shaped. Aksoy's method, on the other hand, required slightly more interactions when using a quadratic utility function and when applied to a linear shaped efficient frontier.

Table 4.1 AVERAGE NUMBER OF PAIRED COMPARISONS (STANDARD DEVIATION)

Type of Utility Function	ACP Method	Aksoy's Method
Linear	10.61 (8.27)	25.57 (22.03)
Quadratic	19.21 (13.86)	30.15 (15.28)
Fourth Power	22.39 (14.77)	24.80 (12.06)
Exponential	11.09 (11.58)	24.19 (21.26)
Shape of Efficient Frontier		
Linear	12.60 (7.83)	31.05 (28.98)
Concave	10.65 (4.43)	27.48 (16.90)
Convex	25.32 (20.63)	25.52 (13.96)
S-Shaped	14.23 (11.09)	25.00 (13.91)
Reverse S	16.33 (11.69)	21.83 (10.60)
Grand Average	15.83 (13.39)	26.18 (18.29)

Although not evident from table 4.1, both methods tended to require fewer interactions with the DM when the variability of the efficient frontier was greater. This is most likely because the data sets with higher variability about the frontier shape tend to have fewer efficient solutions. The maximum number of interactions required with the ACP method was 86, which occurred when using the first, fourth power utility function (coefficients of 5 and 5) and a convex efficient frontier with a variance of one. The ACP method found the best compromise solution with only one paired comparison on seven separate occasions. Aksoy's method required a maximum of 170 paired comparisons, when using the first linear utility function and a linear efficient frontier with variance of one. The least number of interactions required was 9, which occurred 5 separate times.

4.4.2 Number of MP Problems Solved

Another important criterion is the number of single objective MP problems which must be solved before obtaining a best compromise solution to the bicriterion problem. Both methods require the solution of at least one nonlinear MP problem at each iteration, unless the two objective functions are linear, in which case an LP problem must be solved. This criterion is directly proportional to the amount of computation time which would be required in actual applications.

The ACP method requires MP problems in the form of the P_b problems described in chapter 3. To located an incumbent solution's AFNS, the first P_b problem must be solved. If there is no solution, the second P_b problem is attempted. The program counts each P_b problem as one MP problem, even if no solution is found.

Aksoy's branch and bound method requires at least one MP problem to determine the boundaries of each node. For example, to locate a boundary of a node N^{2k-1}, the algorithm must maximize $f_1(x)$ subject to a constraint on $f_2(x)$. If there is more than one solution to this problem, then a second MP problem must be solved which maximizes $f_2(x)$ over the solution space of the first problem. However, for this comparative study, the solution spaces were randomly generated using real numbers, and there are no duplications of either $f_1(x)$ or $f_2(x)$ in the data sets used. Therefore, only one MP problem was counted during testing when determining the boundaries of the nodes. In actual applications, there could very well be more than one solution for each MP problem, resulting in the need for a second MP problem. In other words, the results obtained from this study of Aksoy's method represent the minimum number of MP problems which would be solved for an actual bicriterion problem of similar size. The maximum number of MP problems required would be twice the number shown here, and would occur if there were always multiple solutions to these MP problems.

The results of this comparative study indicate that Aksoy's method required the solution of fewer single objective MP problems. Overall, the ACP method averaged 26.91 MP problems solved before locating the best compromise solution, with a standard deviation of 24.22. Aksoy's method required an average of 20.11 MP problems, with a standard deviation of 10.78.

A comparison of the number of MP problems required according to the shape of the efficient frontier and the utility function used is presented in table 4.2. Aksoy's method requires fewer MP problems for all utility functions except linear, in which the ACP method performed slightly better than Aksoy's. The ACP method solved fewer MP problems when the efficient frontier was linear or concave shaped. It performed poorly with a convex shaped efficient frontier. There was more variation in the number of MP problems required for the ACP method (overall standard deviation of 24.22) than for Aksoy's method (standard deviation of 10.78). The maximum number of MP problems required for the ACP method was 165, which occurred using the second quadratic and first exponential utility functions, both over the convex efficient frontier with a variance of one. The least number of MP solutions required was 5, which occurred on seven different runs. Aksoy's method required a maximum of 106 MP problems, when using the first linear utility function over the linear efficient frontier with variance one. A minimum of 8 MP problems were required using Aksoy's method, which occurred five times. For either method, the maximum number of MP problems required would probably be prohibitively high in solving an actual bicriterion problem of this magnitude.

Table 4.2 AVERAGE NUMBER OF MP PROBLEMS SOLVED (STANDARD DEVIATION)

Type of Utility Function	ACP Method	Aksoy's Method
Linear	18.37 (14.40)	19.76 (13.15)
Quadratic	31.04 (24.26)	22.88 (9.44)
Fourth Power	34.72 (23.49)	19.20 (7.07)
Exponential	20.09 (22.35)	18.61 (11.93)
Shape of Efficient Frontier		
Linear	20.61 (11.43)	22.97 (16.52)
Concave	16.75 (7.08)	20.77 (10.82)
Convex	47.63 (37.45)	19.07 (7.89)
S-Shaped	21.69 (18.60)	19.70 (8.69)
Reverse S	27.89 (20.06)	18.07 (6.29)
Grand Average	26.91 (24.22)	20.11 (10.78)

As previously described, the ACP method always solves the P_b-1 problem first, then if there are no feasible solutions it solves the P_b-2 problem. However, the algorithm would also converge to the best compromise solution if the P_b-2 problem were always attempted first. Therefore, it may be possible to reduce the number of MP problems solved by reversing the order of those problems. Because the highest number of MP problems was required using a quadratic utility function and a convex efficient frontier with small variance, those particular problems were rerun using the ACP method with the P_b problems reversed.

Although the number of MP problems was reduced for the second quadratic utility function (coefficients of 8 and 2), the number of MP problems increased when using the third quadratic function (coefficients of 2 and 8). These results are presented in table 4.2.1, which lists both the number of pairwise comparisons and the number of MP problems solved. Of course, more extensive testing would be required before generalizing any conclusions. Also, in actual applications, the DM's utility function and the shape of the efficient frontier would normally not be known.

Table 4.2.1 NUMBER OF PAIRED COMPARISONS/NUMBER OF MP PROBLEMS SOLVED USING Pb-2 FIRST

Quadratic Utility Fct Convex Frontier	Using P_b-1 First	Using P_b-2 First
(1) Coefficients 5, 5 Frontier Variance 1	77 / 110	79 / 114
(1) Coefficients 5, 5 Frontier Variance 3	34 / 57	41 / 55
(2) Coefficients 8, 2 Frontier Variance 1	82 / 165	83 / 87
(2) Coefficients 8, 2 Frontier Variance 3	30 / 61	30 / 35
(3) Coefficients 2, 8 Frontier Variance 1	51 / 58	50 / 93
(3) Coefficients 2, 8 Frontier Variance 3	24 / 31	25 / 49

4.4.3 Average Utility Function Difference (AUFD) Between Two Solutions Compared by the DM at Each Iteration

The Average Utility Function Difference (AUFD) between each pair of solutions presented to the DM is another indicator of the cognitive burden imposed upon the DM. It is easier for the DM to specify her preference between two solutions when there is a greater difference between the utility function values of those two solutions. This criterion was calculated by averaging the absolute value of the difference between the utility function values of the two solutions at each iteration. The average value for an entire bicriterion problem was output by the program. The data presented in table 4.3 represents the grand mean of this criterion, or the average of the averages for each run. Because the different utility function values were not scaled, the results are separated according to the utility function utilized by the program.

Table 4.3 AVERAGE UTILITY FUNCTION DIFFERENCE BETWEEN TWO SOLUTIONS COMPARED AT EACH ITERATION

Utility Function Frontier Shape	ACP Method	Aksoy's Method
Linear (Cum)	96.19	115.28 *
Linear	55.51	113.95 *
Concave	39.10	83.94 *
Convex	216.87 *	151.10
S-Shaped	87.79	109.47 *
Reverse S	81.69	117.92 *
Quadratic (Cum)	9,651.30 *	6,986.27
Linear	6,799.03 *	6,548.44
Concave	4,144.07	5,409.76 *
Convex	16,080.47 *	6,931.55
S-Shaped	9,405.17 *	8,716.13
Reverse S	11,827.75 *	7,325.47
Fourth Power (Cum)	1.183×10^8 *	8.130×10^7
Linear	7.378×10^7 *	7.268×10^7
Concave	4.585×10^7	5.026×10^7 *
Convex	9.962×10^7 *	7.776×10^7
S-Shaped	1.900×10^8 *	1.101×10^8
Reverse S	1.823×10^8 *	9.575×10^7
Exponential (Cum)	0.1150	0.1390 *
Linear	0.0678	0.1357 *
Concave	0.0477	0.1080 *
Convex	0.2639 *	0.1787
S-Shaped	0.1050	0.1230 *
Reverse S	0.0906	0.1499 *

* The preferred method for the comparison.

The results based on this criterion were pretty evenly split between the two methods. The ACP method has a higher AUFD between two compared solutions when utilizing a quadratic or a fourth power utility function. Aksoy's branch and bound method produces better results for this criterion when using a linear or exponential utility function. However, the ACP method always had a higher value when the efficient frontier was convex, and Aksoy's method always had a higher value when the efficient frontier was concave.

4.4.4 Average Utility Function Difference (AUFD) Between the Ideal Solution and the Preferred Solution at Each Iteration

This criterion compared the difference between the utility function value for a data set's ideal solution, where the ideal solution = [Max $f_1(x)$, Max $f_2(x)$], to the DM's preferred solution from each pairwise comparison. When using a linear utility function, the following formula was used:

$$\text{UFD} = \frac{|\,\text{U(ideal solution)} - \text{U(preferred solution)}\,|}{|\,\text{U(ideal solution)}\,|}$$

However, for the other utility functions, the UFD between the ideal and preferred solutions was not scaled, because the utility function of the ideal solution equaled zero for the quadratic and fourth power utility functions. The AUFD between the ideal and preferred solutions was calculated for each problem. The average value of these averages are presented in table 4.4. Again, the figures are separated according to the utility function used because most of the data is not scaled. A smaller value for this criterion is preferred, since the DM is trying to achieve the ideal solutions, if possible. However, for some iterations Aksoy's method presents the DM with an infeasible solution which is the ideal solution of a current node. Therefore, this criterion is biased slightly in favor of Aksoy's method. Also, if an infeasible solution is presented and the DM were to terminate Aksoy's method before reaching a best compromise solution, the current incumbent would have to serve as the current best solution, even if the node's ideal solution were preferred to it.

The results indicate that in general, the values of solutions presented to the DM using Aksoy's branch and bound method are closer to the ideal solution value than those presented to the DM using the ACP method. When the DM's utility function is linear, the AUFD values using the ACP method are 8.49% higher than the values using Aksoy's method, meaning that the values of the solutions presented to the DM are farther away from the ideal solution value. For quadratic utility functions, the ACP values are 5.58% higher, and ACP values are 19.72% and 19.76% higher using fourth power and exponential utility functions,

Table 4.4 AVERAGE UTILITY FUNCTION DIFFERENCE BETWEEN IDEAL AND PREFERRED SOLUTIONS AT EACH ITERATION

Utility Function Frontier Shape	ACP Method	Aksoy's Method
Linear (Cum)	0.2837	0.2615 *
Linear	0.3120	0.2737 *
Concave	0.2745	0.2395 *
Convex	0.2962	0.2885 *
S-Shaped	0.2630	0.2496 *
Reverse S	0.2727	0.2562 *
Quadratic (Cum)	18,316.92	17,349.15 *
Linear	19,510.36	18,125.80 *
Concave	14,433.01	11,851.75 *
Convex	25,542.61	24,404.85 *
S-Shaped	14,939.94 *	14,993.36
Reverse S	17,158.69 *	17,370.01
Fourth Power (Cum)	7.708×10^7	6.441×10^7
Linear	7.783×10^7	6.253×10^7 *
Concave	4.413×10^7	3.278×10^7 *
Convex	1.711×10^8	1.166×10^8 *
S-Shaped	3.623×10^7 *	4.733×10^7
Reverse S	5.616×10^7 *	6.287×10^7
Exponential (Cum)	0.2327	0.1943 *
Linear	0.2578	0.2030 *
Concave	0.2362	0.1759 *
Convex	0.2319	0.2198 *
S-Shaped	0.2037	0.1820 *
Reverse S	0.2338	0.1909 *

* The preferred method for the comparison.

respectively. In only two cases, when the utility function was quadratic and the shape of the efficient frontier was S-shaped or reverse S-shaped, was the AUFD between the ideal and preferred solutions significantly less for the ACP method than Aksoy's method. Also, when the DM's utility function was linear or exponential, the standard deviation for this criterion was nearly equal between the two methods. However, the ACP method had more variability than Aksoy's method when utilizing a quadratic or fourth power utility function.

4.5 Summary of Comparative Study

This study compared the ACP method with Aksoy's branch and bound method, based on the four criteria described in the previous sections. The methods were compared using four different types of utility functions to represent the DM's preference structure, with each type of function having three variations. The methods were applied to 25 different sets of solutions, based on five different shapes of the efficient frontier with five variances each.

The ACP method required fewer pairwise comparisons than Aksoy's method for all utility functions and for all shapes of the efficient frontier. This is an important result since this criterion is a major indicator of cognitive burden to the DM.

However, Aksoy's method generally performed better than the ACP method when considering the average number of single objective MP problems solved. Overall, Aksoy's method required an average of 20.11 MP problems while the ACP method required an average of 26.91. The ACP method also had greater variability. The ACP method did outperform Aksoy's method on this criterion when the utility function was linear or when the efficient frontier was linear or concave shaped. Also, Aksoy's method would require solving additional MP problems in actual applications if there were any multiple solutions to the MP problems solved.

On the criterion of Average Utility Function Difference (AUFD) between the two solutions compared by the DM at any iteration, there was no significant difference between the two methods. A larger value of this criterion would indicate that the values of two solutions were farther apart and thus it would be less difficult for the DM to specify her preference. The ACP method resulted in a larger average AUFD between two solutions when the utility function was quadratic or fourth power, and Aksoy's method resulted in larger values when the utility function was linear or exponential. One exception was that the ACP method always performed better than Aksoy's method when the efficient frontier was convex, and Aksoy's always performed better when the efficient frontier was concave.

The fourth criterion measured was the AUFD between the ideal solution and the preferred solution at any iteration. The values of the preferred solutions were generally closer to the value of the ideal solution when applying Aksoy's method. However, this is somewhat biased because Aksoy's method allows the DM to select the ideal solution of a current node, which is infeasible, as a preferred solution.

CHAPTER V

CONCLUSIONS AND RECOMMENDATIONS

5.1 Conclusions

In this research, an interactive method for Bicriterion Integer Mathematical Programming (BIMP) problems was presented. This method, called the Artificial Cutting Plane (ACP) method, consists of four major steps:

(1) Determine initial boundaries by separately maximizing the two objective functions. If these two problems result in the same solution, it is optimal and the algorithm terminates. Otherwise, set initial boundaries and let one solution be the current incumbent solution for comparison.

(2) Locate an Associated Frontier Nondominated Solution (AFNS) to the incumbent solution. If there is no AFNS in the remaining feasible area, the current incumbent is the best compromise solution and the algorithm terminates.

(3) Present the DM with the incumbent and its AFNS, and update the constraints on the objective space based on which solution is preferred by the DM.

(4) Let the preferred solution be the new incumbent, and return to step 2.

This method has a number of advantages. It is easy to understand and implement because it exploits the relative simplicity of bicriterion, integer programming problems. It only requires the DM to make pairwise comparisons; that is, the DM need not know specific tradeoff values. Also, the algorithm only presents feasible solutions to the DM. Since the DM need not consider any infeasible solutions, she will not be misled as to the shape of the efficient frontier, and she could terminate the method at any time if she is satisfied with a current, feasible solution. Finally, the assumptions of the algorithm are not overly restrictive, such as requiring concave objective functions and a quasiconcave, increasing utility function.

The ACP method was programmed in Turbo Pascal and compared to Aksoy's branch and bound method. Both methods were used to locate a best compromise solution from 25 data sets of randomly generated solutions. These data sets represented five different shapes of efficient frontiers, each with variance of 1, 3, 5, 7 and 9. To simulate an actual DM's preference, four different types of utility functions were used (linear, quadratic, fourth power and exponential), with three variations of each function.

Four evaluation criteria were used to compare the two methods. The ACP method always required fewer interactions with the DM, which is an important measure of the cognitive burden to the DM. However, on the average, Aksoy's method required the solution of fewer single objective MP problems. The figures for Aksoy's method represent the minimum number of MP problems required, since additional problems would be required if there were multiple solutions to any problem. The ACP method did require the solution of fewer MP problems when the utility function was linear or when the efficient frontier was linear or concave shaped. The two methods were quite competitive with respect to the Average Utility Function Difference (AUFD) between the two solutions presented to the DM at any iteration. Finally, Aksoy's method showed better results for the AUFD between the ideal solution and the preferred solution from each iteration. However, this considers the iterations of Aksoy's method when an infeasible solution is preferred by the DM.

Because the ACP method required significantly fewer interactions with the DM than Aksoy's method, and resulted in roughly equal values of the AUFD between two solutions compared by the DM, I would conclude that the ACP method imposes less of a cognitive burden on the DM. However, Aksoy's method usually required the solution of fewer single objective MP problems, which means that it would require less computation time than the ACP method. Based upon the results of this study, the ACP method generated very competitive results against Aksoy's branch and bound method.

The major contribution of this research is the development of an algorithm to solve Bicriterion, Integer

Mathematical Programming (BIMP) problems. Another contribution is a comparative study of the ACP method with Aksoy's branch and bound method applied to pure integer bicriterion problems. This study illustrates a procedure for comparative studies without an actual DM and without actually solving one or more single objective mathematical programming problems at each iteration.

5.2 Recommendations

There are some interesting aspects of this research that have not yet been fully explored. Some of these aspects are as follows:

(1) Exploring ways to reduce the number of MP problems required by the ACP method, such as by changing the order of the P_b problems under certain circumstances.

(2) Conducting a comparative study using an actual bicriterion problem with an actual DM.

(3) Extending this algorithm to mixed integer bicriterion problems.

(2) Extending this algorithm to multiple objective optimization.

REFERENCES

Adulbhan, Pakorn and Mario T. Tabucanon. 1977. Bicriterion Linear Programming. Computing and Operations Research 4: 147-153.

Aksoy, Yasemin. 1990. An Interactive Branch-and-Bound Algorithm for Bicriterion Nonconvex/Mixed Integer Programming. Naval Research Logistics 37: 403-417.

Aneja, Y. P. and K. P. K. Nair. 1979. Bi-criteria Transportation Problem. Management Science 25 (1): 73-78.

Bacopoulos, A. and I. Singer. 1977. On Convex Vectorial Optimization in Linear Spaces. Journal of Optimization Theory and Applications 21 (February): 175-188.

Balinski, M. L., ed. 1974. Approaches to Integer Programming. Amsterdam: North-Holland Publishing Company.

Benson, H. P. 1979. Vector Maximization with Two Objective Functions. Journal of Optimization Theory and Applications 28 (June): 253-257.

Boyle, Carolyn Redding. 1992. A Paired-Comparison/Cutting-Plane Approach to Optimizing Multiple-Response Simulation Experiments. Ph.D. dissertation, Mississippi State University.

Choo, E. U. and D. R. Atkins. 1982. Bicriteria Linear Fractional Programming. Journal of Optimization Theory and Applications 36 (February): 203-220.

Chalmet, L. G., L. LeMondis and D. J. Elzinga. 1986. An Algorithm for the Bi-criterion Integer Programming Problem. European Journal of Operational Research 25: 292-300.

Climaco, J. C. N. and E. Q. V. Martins. 1982. A Bicriterion Shortest Path Algorithm. European Journal of Operational Research 11: 399-404.

Cochrane, James L. and Milan Zeleny, ed. 1973. Multiple Criteria Decision Making. Columbia, SC: University of South Carolina Press.

Cohon, Jared L., Richard R. Church and Daniel P. Sheer. 1979. Generating Multiobjective Trade-Offs: An Algorithm for Bicriterion Problems. Water Resources Research 15 (October): 1001-1010.

Dagpunar, John. 1988. Principles of Random Variate Generation. New York: Oxford University Press.

Eswaran, P. K., A. Ravindran and H. Moskowitz. 1989. Algorithms for Nonlinear Integer Bicriterion Problems. Journal of Optimization Theory and Applications 23 (November): 261-279.

Eswaran, P. K. 1983. Interactive Decision Making Involving Bicriterion Problems - Algorithms and Application. Ph.D. diss., Purdue University.

Evans, G. W. 1984. An Overview of Techniques for Solving Multiobjective Mathematical Programs. Management Science 30: 1268-1282.

French, S., et al, ed. 1983. Multi-Objective Decision Making. New York: Academic Press, Inc.

French, Simon. 1986. Decision Theory: An Introduction to the Mathematics of Rationality. Chichester, England: Ellis Horwood, Ltd.

Geoffrion, Arthur M. 1967. Solving Bicriterion Mathematical Programs. Operations Research 15: 39-54.

Henig, Mordechai I. 1985. The Shortest Path Problem with Two Objective Functions. European Journal of Operational Research 25: 281-291.

Hillier, Frederick S., and Gerald J. Lieberman. 1980 Introduction to Operations Research. 3rd ed. Oakland, CA: Holden-Day, Inc.

John, Thuruthickara C. and Randall P. Sadowski. 1984. On a Bicriteria Scheduling Problem. Presented at TIMS/ORSA Joint National Conference, Dallas, November.

Kiziltan, Gulseren and Erkut Yucaoglu. 1982. An Algorithm for Bicriterion Linear Programming. European Journal of Operational Research 10: 406-411.

Korhonen, Pekka, Jyrki Wallenius and Stanley Zionts. 1984. Solving the Discrete Multiple Criteria Problem Using Convex Cones. Management Science 30 (11): 1336-1345.

Malhotra, R. 1982. A Note on Bi-Criteria Transportation Problems. Zeitschrift fur Operations Research 26: 259-261.

Pasternak, H., and U. Passy. 1973. Bicriterion Mathematical Programs with Boolean Variables. In Multiple Criteria Decision Making, ed. James L. Cochrane and Milan Zeleny, 327-348. Columbia, SC: University of South Carolina Press.

Ramachandran, Aravinda. 1989. An Interactive Bicriterion Solution Method and Its Application to CPM Problems. M.S. Thesis, Mississippi State University.

Reklaitis, G. V., A. Ravindran and K. M. Ragsdell. 1983. Engineering Optimization Method and Applications. New York: John Wiley & Sons.

Sadagopan, S. and A. Ravindran. 1982. Interactive Solution of Bi-Criteria Mathematical Programs. Naval Research Logistics Quarterly 29 (September): 443-459.

Steuer, Ralph E. 1986. Multiple Criteria Optimization: Theory, Computation, and Application. New York: John Wiley & Sons.

Shin, Wan S. and A. Ravindran. 1991. Interactive Multiple Objective Optimization: Survey I - Continuous Case. Computers Operations Research 18 (1): 97-114.

Shin, Wan S. and A. Ravindran. A comparative study of interactive tradeoff cutting plane methods for MOMP. Forthcoming in European Journal of Oparations Research.

Walker, John. 1978. An Interactive Method as an Aid in Solving Bicriterion Mathematical Programming Problems. Journal of the Operational Research Society 29 (9): 915-922.

Zionts, Stanley. 1981. A Multiple Criteria Method for Choosing Among Discrete Alternatives. European Journal of Operational Research 7: 143-147.

Zionts, Stanley and Jyrki Wallenius. 1980. Identifying Efficient Vectors: Some Theory and Computational Results. Operations Research 28 (3): 785-793.

APPENDIX A

GENERATING SOLUTIONS

PROGRAM LISTING

```
PROGRAM GEN_SOLNS;


CONST

  NUM = 500; { number of solutions to generate   }
  MAX = 100; { range of f1 will be from 0 to MAX }
  STD_DEV = 9;     { standard deviation of f2(x)  }
  FILE_NAME: STRING[12] = 'S9.SOL'; { output file }

VAR

  POINT :  ARRAY[1..NUM, 1..2] OF REAL;


PROCEDURE GEN_F1;  { generates NUM random values of }
                   { f1(x), between 0 and MAX       }

VAR

  I : INTEGER;


BEGIN
  RANDOMIZE;
  FOR I := 1 TO NUM DO
    POINT[I,1] := MAX * RANDOM;

END;  {GEN_F1}
```

```
PROCEDURE GEN_F2;
  { Generates a value of f2(x) for each value    }
  { of f1(x).  f2(x) is normally distributed     }
  { about a specified shape of the efficient     }
  { frontier.  Mean of f2 = -.0004(f1-50)^3 + 50 }

VAR

  I, J : INTEGER;
  K, RV, MEAN, B, S, U1, U2, SIGN : REAL;
  STORED:  BOOLEAN;
    { whether or not an RV is already stored }

BEGIN

  STORED := FALSE;
  FOR I := 1 TO NUM DO
    BEGIN
    { generate a standard Normal deviate using }
    { polar Box-Muller method:                 }
    IF STORED = TRUE THEN RV := B * U2 ELSE
      BEGIN
      REPEAT
        U1 := 2 * RANDOM - 1;
        U2 := 2 * RANDOM - 1;
        S := U1 * U1 + U2 * U2;
      UNTIL S < 1;
      B := SQRT((-2 * LN(S))/S);
      RV := B * U1;
      END;
    IF STORED = FALSE THEN STORED := TRUE
      ELSE STORED := FALSE;

    { convert standard Normal to Normal with }
    { specified mean and variation:          }
    SIGN := ABS(POINT[I,1] - 50)/(POINT[I,1] - 50);
    MEAN := -0.0004 * SIGN * EXP(3 *
            LN(ABS(POINT[I,1] - 50))) + 50;
    POINT[I,2] := RV * STD_DEV + MEAN;
    END;

END;  {GEN_F2}
```

```
PROCEDURE SEND_TO_FILE;  { send results to output file }

VAR

  I : INTEGER;
  FILE_VAR : FILE OF REAL;

BEGIN

  ASSIGN(FILE_VAR, FILE_NAME);
  REWRITE(FILE_VAR);  { opens file named FILE_NAME }
  FOR I := 1 TO NUM DO
    BEGIN
    WRITE(FILE_VAR, POINT[I,1]);
    WRITE(FILE_VAR, POINT[I,2]);
    END;
  CLOSE(FILE_VAR);

END;  {SEND_TO_FILE}


BEGIN  {MAIN PROGRAM}

GEN_F1;
GEN_F2;
SEND_TO_FILE;

END.
```

APPENDIX B

ARTIFICIAL CUTTING PLANE (ACP)

PROGRAM LISTING

```
(*******************************************************)
(*                                                     *)
(*  ARTIFICIAL CUTTING PLANE METHOD                    *)
(*                                                     *)
(*  WRITTEN BY:  Diane Breivik Allen                   *)
(*                                                     *)
(*  Used to determine best compromise solution         *)
(*  after reading in a file of bicriterion solutions.  *)
(*                                                     *)
(*******************************************************)

PROGRAM ALLEN;

USES CRT, PRINTER;

CONST
  MAX_DATA = 500; { number of solutions entered }
  VSN = 0.000001;
  FILE_NAME : STRING[12] = 'LIN5.SOL';
  UTIL_NAME : STRING[12] = 'LINEAR(2,8)';

VAR
  POINT:  ARRAY[1..MAX_DATA, 1..2] OF REAL;
    { holds solutions }
  SOLN:  BOOLEAN;
    { records whether there is a sol'n to Pb problems }
  OPTIMAL:  BOOLEAN;
    { true if there is a single, optimal solution }
  MAX1, MAX2:  REAL;
    { holds actual maximum values of f1(x) and f2(x) }
  MAX_POSN1, MAX_POSN2:  INTEGER;
    { holds place number of array POINT }
    { where maximum values are found    }
  LEFT, RIGHT:  INTEGER;
    { holds place of current LEFT and RIGHT       }
    { solutions; used in Pb problems and COMPARE }
  MIN1, MIN2:  REAL;
    { values of current constraints on f1(x) and f2(x) }
  COUNT,
    { keeps track of the number of paired comparisons  }
  COUNT_SOL: INTEGER;
    { counts number of solutions generated/ solved for }
  IDEALDIF,
    { records value difference between preferred solu- }
    { tion and ideal solution; records value function  }
  FUNCDIF:  ARRAY[1..MAX_DATA] OF REAL;
    { difference between compared pts }
```

```
PROCEDURE GET_SOLNS;  { reads solutions from file }

VAR
  FILE_VAR:  FILE OF REAL;
  I: INTEGER;

BEGIN
  ASSIGN(FILE_VAR, FILE_NAME);
  RESET(FILE_VAR);
  FOR I := 1 TO MAX_DATA DO
    BEGIN
    READ(FILE_VAR, POINT[I,1]);
    READ(FILE_VAR, POINT[I,2]);
    END;
  CLOSE(FILE_VAR);
END;  {GET_SOLNS}


PROCEDURE FIND_MAX(J: INTEGER; VAR MAX: REAL;
                   VAR MAX_POSN: INTEGER);

 { Will pass J = 1 when maximizing f1(x); J = 2 when  }
 { maximizing f2(x).  MAX_POSN holds the position in  }
 { array POINT of the current maximum function value. }
 { MAX holds the actual current maximum value.        }

  VAR
    I,            { counter }
    K:  INTEGER;  { index of fc't not being maximized }

  BEGIN
  FOR I := 2 TO MAX_DATA DO
    BEGIN
    IF POINT[I,J] > MAX THEN
      BEGIN
      MAX := POINT[I,J];
      MAX_POSN := I;
      END;
    IF POINT[I,J] = MAX THEN  { Break tie by comparing }
        { value of other function.  If both function   }
        { values are equal, it doesn't matter          }
        { which point is chosen for this program.      }
      BEGIN
      IF (J = 1) THEN K := 2 ELSE K := 1;
      IF POINT[I,K] > POINT[MAX_POSN,K] THEN
        MAX_POSN := I;
      END;
    END;  {FOR I statement }
  END;   {FIND_MAX}
```

```
PROCEDURE Pb_RIGHT;
   { Finds AFNS by swinging right of point 1 (LEFT) }

  VAR
    I:  INTEGER;
    B, BMAX:  REAL;

  BEGIN
  SOLN := FALSE;  { initialize }
  BMAX := 0;
  FOR I := 1 TO MAX_DATA DO
    IF POINT[I,1] > POINT[LEFT,1] THEN
    { only checks points to the right  }
      IF POINT[I,2] > MIN2 THEN
      { only checks points that meet constraint }
        BEGIN
        B := (POINT[I,1] - POINT[LEFT,1]) /
             (POINT[LEFT,2] - POINT[I,2]);
        IF B > BMAX THEN
        { update BMAX and new RIGHT comparison point }
          BEGIN
          BMAX := B;
          RIGHT := I;
          END;
        IF B = BMAX THEN  { more than 1 point on line }
          IF POINT[I,1] > POINT[RIGHT,1] THEN
          { only update BMAX if the new point is       }
          { farthest away and thus the AEEP (arbitrary)}
            BEGIN
            BMAX := B;
            RIGHT := I;
            END;
        SOLN := TRUE;
        { a solution has been found to this Pb problem }
        END;  {IF LOOP }
  COUNT_SOL := COUNT_SOL + 1;
  { one MP problem is solved in this step }

  END;  {Pb_RIGHT}
```

```
PROCEDURE Pb_LEFT;
   { Finds AFNS by swinging left of point r (RIGHT) }

  VAR
    B, BMAX: REAL;
    I: INTEGER;

  BEGIN
  RIGHT := LEFT;
    { The current "lefthand" point is now considered }
    { the "righthand" point.  This procedure is only }
    { called when there are no points to the right.  }
  SOLN := FALSE;  { initialize }
  BMAX := 0;
  FOR I := 1 TO MAX_DATA DO
    { only checks points to the left of RIGHT and }
    { points that meet current constraints        }
    IF (POINT[I,1] < POINT[RIGHT,1]) AND (POINT[I,2] >
      POINT[RIGHT,2]) AND (POINT[I,1] > MIN1)  THEN
        BEGIN
        B :=  (POINT[RIGHT,2] - POINT[I,2]) /
              (POINT[I,1] - POINT[RIGHT,1]);
        IF B > BMAX THEN
        { update BMAX and new LEFT comparison point }
          BEGIN
          BMAX := B;
          LEFT := I;
          END;
        IF B = BMAX THEN  { more than 1 point on line }
          IF POINT[I,1] < POINT[LEFT,1] THEN
            BEGIN
            BMAX := B;
            LEFT := I;
            END;
        SOLN := TRUE;
        { a solution has been found to this Pb problem }
        END;  {IF LOOP }
  COUNT_SOL := COUNT_SOL + 1;
  { one MP problem is solved in this step }

  END;  {Pb_LEFT}
```

```
PROCEDURE COMPARE;
   { Compares two solutions -- Equivalent to asking DM }
   { which solution is preferred.  This procedure also }
   { updates the constraints and l and r, if necessary }

  VAR
    VAL_LEFT, VAL_RIGHT, VAL_IDEAL:  REAL;


    FUNCTION U(F: INTEGER):REAL;  { utility function }
      BEGIN

      U := 2*POINT[F,1] + 8*POINT[F,2];

      END;

  BEGIN {COMPARE}

  VAL_IDEAL := 2*MAX1 + 8*MAX2;
  COUNT := COUNT + 1;
  { counts number of paired comparisons }
  VAL_LEFT := U(LEFT);
  VAL_RIGHT := U(RIGHT);
  IF VAL_LEFT >= VAL_RIGHT THEN
    BEGIN
    MIN2 := POINT[RIGHT,2];
    { value difference between preferred & ideal:   }
    IDEALDIF[COUNT] := ABS(VAL_IDEAL - VAL_LEFT)
                       /ABS(VAL_IDEAL);
    END;
  IF VAL_RIGHT >= VAL_LEFT THEN
    BEGIN
    MIN1 := POINT[LEFT,1];
    LEFT := RIGHT;  { new "incumbent" to begin Pb right }
    IDEALDIF[COUNT] := ABS(VAL_IDEAL - VAL_RIGHT)
                       /ABS(VAL_IDEAL);
    END;

  { value difference between two solutions compared:   }
  FUNCDIF[COUNT] := ABS(VAL_LEFT - VAL_RIGHT);

  END;  {COMPARE}
```

```
PROCEDURE RESULTS;   {generates output}

VAR
  I,J,LIMIT:  INTEGER;
  TOT_VDI, TOT_VDF, TOT2VDI, TOT2VDF, AVG_VDI, AVG_VDF,
  SD_VDI, SD_VDF:  REAL;  { used in calculation of mean & }
                          { std dev of value differences   }
  BEGIN
  WRITELN(LST);
  WRITELN(LST);
  WRITELN(LST);
  WRITELN(LST);
  WRITELN(LST,'   This run used data set ',FILE_NAME);
  WRITELN(LST,'   and utility function ',UTIL_NAME);
  WRITELN(LST);
  IF OPTIMAL THEN
    BEGIN
    WRITELN(LST,'   This problem has a single, optimal
                    solution which maximizes ');
    WRITELN(LST,'   both objective functions.  No
                    comparisons were necessary. ' );
    WRITELN(LST);
    WRITELN(LST,'   The solution to this problem is
                    f1(x) = ',MAX1:5:4,' and ');
    WRITELN(LST,'   f2(x) = ',MAX2:5:4);
    END
  ELSE
    BEGIN
    WRITELN(LST,'   The solution to this problem is
                    f1(x) = ',POINT[RIGHT,1]:5:4);
    WRITELN(LST,'   and f2(x) = ',POINT[RIGHT,2]:5:4);
    WRITELN(LST);
    WRITELN(LST,'   It was found after ',COUNT,' paired
                    comparisons. ');
    WRITELN(LST);
    WRITELN(LST,'   ',COUNT_SOL,' MP problems were
                    solved.');
    END;
  WRITELN(LST);
  WRITELN(LST,'   The ideal point was ',MAX1:5:4,' ',
                  MAX2:5:4);
```

```
{ calculate average value function differences and }
{ standard deviations                               }
TOT_VDI := 0;
TOT_VDF := 0;
FOR I := 1 TO COUNT DO
  BEGIN
  TOT_VDI := TOT_VDI + IDEALDIF[I];
  TOT_VDF := TOT_VDF + FUNCDIF[I];
  END;
AVG_VDI := TOT_VDI/COUNT;
AVG_VDF := TOT_VDF/COUNT;
TOT2VDI := 0;
TOT2VDF := 0;
FOR I := 1 TO COUNT DO
  BEGIN
  TOT2VDI := TOT2VDI + SQR(IDEALDIF[I] - AVG_VDI);
  TOT2VDF := TOT2VDF + SQR(FUNCDIF[I] - AVG_VDF);
  END;
IF COUNT = 1 THEN
  BEGIN
  SD_VDI := 0;
  SD_VDF := 0;
  END
ELSE
  BEGIN
  SD_VDI := SQRT(TOT2VDI/(COUNT-1));
  SD_VDF := SQRT(TOT2VDF/(COUNT-1));
  END;
WRITELN(LST,'   The average value difference between
                the most preferred point ');
WRITELN(LST,'   at any iteration and the ideal point
                is ', AVG_VDI:5:5);
WRITELN(LST,'   The standard deviation is ',
                SD_VDI:5:5);
WRITELN(LST);
WRITELN(LST,'   The average value difference between
                two solutions compared by');
WRITELN(LST,'   the DM at any iteration is ',
                AVG_VDF:5:4);
WRITELN(LST,'   The standard deviation is ',
                SD_VDF:5:4);
WRITELN(LST);
WRITELN('Press any key to continue');
REPEAT UNTIL KEYPRESSED;
WRITELN(LST,CHR(12));  { form feed command }

END;  {RESULTS}
```

```
BEGIN {MAIN PROGRAM}

CLRSCR;
COUNT := 0;  {initialize}
COUNT_SOL := 2;  { to include the two initial boundary }
                 { solutions generated                 }
OPTIMAL := FALSE;

GET_SOLNS;

MAX1 := POINT[1,1];  {initialize}
MAX_POSN1 := 1;
FIND_MAX(1, MAX1, MAX_POSN1);  { maximize f1(x) }

MAX2 := POINT[1,2];
MAX_POSN2 := 1;
FIND_MAX(2, MAX2, MAX_POSN2);  { maximize f2(x) }

LEFT := MAX_POSN2;
MIN1 := POINT[MAX_POSN2,1] - VSN;
MIN2 := POINT[MAX_POSN1,2] - VSN;

IF MAX_POSN1 = MAX_POSN2 THEN OPTIMAL := TRUE;
{ There is a single optimal solution.  }
IF NOT OPTIMAL THEN
  REPEAT

    Pb_RIGHT;
    IF NOT SOLN THEN Pb_LEFT;
    IF SOLN THEN COMPARE;

  UNTIL NOT SOLN;

RESULTS;

END.  {MAIN PROGRAM}
```

APPENDIX C

AKSOY'S METHOD PROGRAM LISTING

```
(*********************************************************)
(*                                                       *)
(*  AKSOY'S BRANCH AND BOUND METHOD                      *)
(*                                                       *)
(*  ALGORITHM BY:  Yasemin Aksoy [1990]                  *)
(*                                                       *)
(*  PROGRAM BY:  Diane Breivik Allen                     *)
(*                                                       *)
(*  Used to determine best compromise solution based     *)
(*  on utility function to represent DM, after read-     *)
(*  ing in a file of bicriterion solutions.              *)
(*                                                       *)
(******************************************************** )


PROGRAM AKSOY;

USES CRT, PRINTER;

LABEL LOOP;

CONST
  MAX_DATA = 500;  { number of solutions entered }
  MAX_NODE = 1000; { maximum number of nodes     }
  FILE_NAME : STRING[12] = 'BACKS9.SOL';
  UTIL_NAME : STRING[12] = 'LINEAR(2,8)';

VAR
  POINT:  ARRAY[1..MAX_DATA, 1..2] OF REAL;
    { holds random solutions   }
  OPTIMAL, { true if there is a single, optimal sol'   }
  EMPTY,   { true when candidate list is empty         }
  BETTER:  BOOLEAN;
    { true if a given point is preferred to incumbent  }
  MAX1, MAX2,
    { holds actual maximum values of f1(x) and f2(x)   }
  Y2,       { current y2 (2k - 1)                      }
  INCUM1,
  INCUM2,
    { actual function values of current incumbent      }
  HALF:  REAL;
    { used to branch nodes - half distance of f2       }
  MAX_POSN1, MAX_POSN2:  INTEGER;
    { holds place number of array POINT   }
    { where maximum values are found      }
  COUNT,
    { keeps track of the number of paired comparisons  }
  COUNT_SOL,
    { counts number of solutions generated (solved for)}
  K,
    { current interation number; used to number nodes  }
  INCUM,
    { place number of POINT where current incumbent is }
  NODE_NUM,       { current node number                }
  L: INTEGER;     { used to increment in main program  }
  CAND_LIST:  ARRAY[0..MAX_NODE] OF INTEGER;
    { refer to place in NODES }
  NODE:  ARRAY[0..MAX_NODE, 1..4] OF REAL;
  L1, U1, L2, U2: REAL;
    { values used to define current node               }
  IDEALDIF,
  FUNCDIF:  ARRAY[1..MAX_NODE] OF REAL;
    { keep track of value difference }


PROCEDURE GET_SOLNS;  { reads solutions from file }

VAR
  FILE_VAR:  FILE OF REAL;
  I: INTEGER;

BEGIN

  ASSIGN(FILE_VAR, FILE_NAME);
  RESET(FILE_VAR);
  FOR I := 1 TO MAX_DATA DO
    BEGIN
    READ(FILE_VAR, POINT[I,1]);
    READ(FILE_VAR, POINT[I,2]);
    END;
  CLOSE(FILE_VAR);

END;  {GET_SOLNS}
```

```
PROCEDURE FIND_MAX(J: INTEGER; VAR MAX: REAL;
                   VAR MAX_POSN: INTEGER);

 { Determines initial "interval of nondominance".       }
 { Will pass J = 1 when maximizing f1(x); J = 2 when    }
 { maximizing f2(x).  MAX_POSN holds the position in    }
 { array POINT of the current maximum function value.   }
 { MAX holds the actual current maximum value.          }

  VAR

    I,                  { counter }
    OTHER:  INTEGER;    { function not being maximized }

  BEGIN

  MAX := POINT[1,J];
  MAX_POSN := 1;
  IF (J = 1) THEN OTHER := 2 ELSE OTHER := 1;
  FOR I := 2 TO MAX_DATA DO
    BEGIN
    IF POINT[I,J] > MAX THEN
      BEGIN
      MAX := POINT[I,J];
      MAX_POSN := I;
      END;
    IF POINT[I,J] = MAX THEN  { Break tie by comparing }
        { value of other function.  If both function   }
        { values are equal, it doesn't matter          }
        { which point is chosen for this program.      }

      IF POINT[I,OTHER] > POINT[MAX_POSN,OTHER] THEN
        MAX_POSN := I;
    END;  {FOR I statement }

  END;   {FIND_MAX}


PROCEDURE STORE_NODE(NUM: INTEGER; L1,U1,L2,U2: REAL);

  BEGIN
  CAND_LIST[NUM] := NUM;
  NODE[NUM, 1] := L1,
  NODE[NUM, 2] := U1;
  NODE[NUM, 3] := L2;
  NODE[NUM, 4] := U2;
  END;
```

```
PROCEDURE GET_NODE;
{ remove node with smallest number from cand list }

  BEGIN
  NODE_NUM := -1;
  REPEAT
    NODE_NUM := NODE_NUM + 1;
  UNTIL CAND_LIST[NODE_NUM] = NODE_NUM;
  CAND_LIST[NODE_NUM] := MAX_DATA + 1;
    { this node no longer in candidate list }
  END;


PROCEDURE CHECK_LIST;
{ checks candidate list to see if it's empty }

VAR
  I: INTEGER;

  BEGIN
  EMPTY := TRUE;    { initialize }
  FOR I := 0 TO MAX_DATA DO
    IF CAND_LIST[I] = I THEN EMPTY := FALSE;
  END;


PROCEDURE COMPARE(TEST1, TEST2: REAL);
   { Pass in acual function values of solution }
   { to compare with incumbent.                }
   { Compares two solutions -- Equivalent to   }
   { asking DM which solution is preferred     }

VAR
  VAL_TEST, VAL_INCUM, VAL_IDEAL:  REAL;

  FUNCTION U(F1, F2: REAL): REAL;  { utility function }

    BEGIN
    U := 2*F1 + 8*F2;
    END;

  BEGIN {COMPARE}

  BETTER := FALSE;      { initialize }
  COUNT := COUNT + 1;
  { counts number of paired comparisons }
  VAL_IDEAL := U(MAX1, MAX2);
  VAL_TEST := U(TEST1, TEST2);
  VAL_INCUM := U(INCUM1, INCUM2);
  IF VAL_TEST > VAL_INCUM THEN
    BEGIN
    BETTER := TRUE;
    IDEALDIF[COUNT] := ABS(VAL_IDEAL - VAL_TEST)
                       /ABS(VAL_IDEAL);
    END
  ELSE IDEALDIF[COUNT] := ABS(VAL_IDEAL - VAL_INCUM)
                          /ABS(VAL_IDEAL);
  FUNCDIF[COUNT] := ABS(VAL_TEST - VAL_INCUM);

  END;


PROCEDURE MAKE_1STNODE;  { create node 2k - 1 }

VAR
  NEW_NODE,       { number of node being created         }
  PLACE,          { holds place value of current max     }
                  { f1(x), given f2(x) > half            }
  I : INTEGER;    { counter used to cycle through array }
  MAX: REAL;      { actual value of current max f1(x)    }

  BEGIN
  NEW_NODE := 2*K - 1;
  COUNT_SOL := COUNT_SOL + 1;  { 1 solution generated }
  MAX := -100;  { initialize }
  FOR I := 1 TO MAX_DATA DO    { max f1 for f2 > half }
    BEGIN
    IF POINT[I,2] >= HALF THEN
      BEGIN
      IF POINT[I,1] > MAX THEN
        BEGIN
        MAX := POINT[I,1];
        PLACE := I;
        END;
      IF POINT[I,1] = MAX THEN
      { if f1 is tied, compare f2 and choose max }
        IF POINT[I,2] > POINT[PLACE,2] THEN
          BEGIN
          MAX := POINT[I,1];
          PLACE := I;
          END;
      END;
    END;
  L1 := NODE[NODE_NUM,1];
        { L1(2K-1) = L1 of current node }
  U1 := MAX;
        { U1 = f1 of point just found = y1 (2k-1) }
  L2 := POINT[PLACE, 2];
        { L2 = f2 of point just found = y2 (2k-1) }
  U2 := NODE[NODE_NUM,4];
        { U2(2k-1) = U2 of current node }
  IF U2 <> L2 THEN
  { points are not equal or on same horizontal line }
    BEGIN
    COMPARE(MAX, POINT[PLACE,2]);
    { compare new point with incumbent }
    IF BETTER THEN  { update incumbent }
      BEGIN
      INCUM1 := MAX;
      INCUM2 := POINT[PLACE,2];
      INCUM := PLACE;
      END;
    STORE_NODE(NEW_NODE, L1, U1, L2, U2);
    { node only stored if it contains more than 1 pt }
    END;

  END;  {MAKE_1STNODE}


PROCEDURE MAKE_2NDNODE;  { create node 2k }

VAR
  NEW_NODE,       { number of node being created        }
  PLACE,          { holds place value of current max    }
                  { f1(x), given f2(x)           > half }
  I : INTEGER;    { counter used to cycle through array }
  MAX : REAL;     { actual value of current max f1(x)   }

  BEGIN
  NEW_NODE := 2*K;
  COUNT_SOL := COUNT_SOL + 1;    { one sol'n generated }
  IF L2 = HALF THEN
    BEGIN
    L1 := U1;
    U1 := NODE[NODE_NUM,2];
        { U1(2k) = U1 of current node r                 }
    L2 := NODE[NODE_NUM,3];
        { L2(2k) = L2 of current node, node_num = r }
    U2 := L2;
    END
```

```
ELSE
  BEGIN
  MAX := -100;  { initialize }
  FOR I := 1 TO MAX_DATA DO
  { maximize f2 for f1 > u1 = y1 (2k-1) }
    BEGIN
    IF POINT[I,1] > U1 THEN
      BEGIN
      IF POINT[I,2] > MAX THEN
        BEGIN
        MAX := POINT[I,2];
        PLACE := I;
        END;
      IF POINT[I,2] = MAX THEN
      { if f2 is tied, compare f1 and choose max }
        IF POINT[I,1] > POINT[PLACE,1] THEN
          BEGIN
          MAX := POINT[I,2];
          PLACE := I;
          END;
      END; {if loop}
    END; {for loop}
  L1 := POINT[PLACE, 1];
        { L1(2K) = y1(2K) or f1 of point just found }
  U1 := NODE[NODE_NUM,2];
        { U1(2k) = U1 of current node r           }
  L2 := NODE[NODE_NUM,3];
        { L2(2k) = L2 of current node, node_num = r }
  U2 := MAX;
        { U2(2k) = y2(2k) or f2 of point just found }
  END;  {ELSE}
IF L1 <> U1 THEN  { points are not equal }
  BEGIN
  IF U2 <> HALF THEN
    BEGIN  { don't compare again if y2(2k) = y2(2k-1)}
    COMPARE(POINT[PLACE,1],MAX);
    IF BETTER THEN  { update incumbent }
      BEGIN
      INCUM1 := POINT[PLACE,1];
      INCUM2 := MAX;
      INCUM := PLACE;
      END;
    END;
  STORE_NODE(NEW_NODE, L1, U1, L2, U2);
  {node only stored if it contains more than 1 pt }
  END;

END;  {MAKE_2NDNODE}
```

```
PROCEDURE RESULTS;   {generates output}

VAR
  I,J,LIMIT:  INTEGER;
  TOT_VDI, TOT_VDF, TOT2VDI, TOT2VDF, AVG_VDI, AVG_VDF,
  SD_VDI, SD_VDF:  REAL;

  BEGIN
  WRITELN(LST);
  WRITELN(LST);
  WRITELN(LST);
  WRITELN(LST);
  WRITELN(LST,'   This run of Aksoy''s method used data
                  set ',FILE_NAME);
  WRITELN(LST,'   and utility function ',UTIL_NAME);
  WRITELN(LST);
  IF OPTIMAL THEN
    BEGIN
    WRITELN(LST,'   This problem has a single, optimal
                    soution which maximizes ');
    WRITELN(LST,'   both objective functions.  No
                    comparisons were necessary. ' );
    WRITELN(LST);
    WRITELN(LST,'   The solution to this problem is
                    f1(x) = ',MAX1:5:4,' and ');
    WRITELN(LST,'   f2(x) = ',MAX2:5:4);
    END
  ELSE  { NOT OPTIMAL }
    BEGIN
    WRITELN(LST,'   The solution to this problem is
                    f1(x) = ',POINT[INCUM,1]:5:4);
    WRITELN(LST,'   and f2(x) = ',POINT[INCUM,2]:5:4);
    WRITELN(LST);
    WRITELN(LST,'   It was found after ',COUNT,' paired
                    comparisons. ');
    WRITELN(LST);
    WRITELN(LST,'   ',COUNT_SOL,' solutions were
                    considered.');
    END;
  WRITELN(LST);
  WRITELN(LST,'   The ideal point was ',MAX1:5:4,' ',
                  MAX2:5:4);
  WRITELN(LST);
```

```
{ calculate average value function differences }
{ and standard deviations                      }
TOT_VDI := 0;
TOT_VDF := 0;
FOR I := 1 TO COUNT DO
  BEGIN
  TOT_VDI := TOT_VDI + IDEALDIF[I];
  TOT_VDF := TOT_VDF + FUNCDIF[I];
  END;
AVG_VDI := TOT_VDI/COUNT;
AVG_VDF := TOT_VDF/COUNT;
TOT2VDI := 0;
TOT2VDF := 0;
FOR I := 1 TO COUNT DO
  BEGIN
  TOT2VDI := TOT2VDI + SQR(IDEALDIF[I] - AVG_VDI);
  TOT2VDF := TOT2VDF + SQR(FUNCDIF[I] - AVG_VDF);
  END;
SD_VDI := SQRT(TOT2VDI/(COUNT-1));
SD_VDF := SQRT(TOT2VDF/(COUNT-1));
WRITELN(LST,'   The average value difference between
                the most preferred point ');
WRITELN(LST,'   at any iteration and the ideal point
                is ', AVG_VDI:5:5);
WRITELN(LST,'   The standard deviation is ',
                SD_VDI:5:5);
WRITELN(LST);
WRITELN(LST,'   The average value difference between
                two solutions compared by');
WRITELN(LST,'   the DM at any iteration is ',
                AVG_VDF:5:4);
WRITELN(LST,'   The standard deviation is ',
                SD_VDF:5:4);
WRITELN(LST);
WRITELN('Press RETURN to continue');
READLN;
WRITELN(LST, CHR(12));  { send paper to next page }

END;  {RESULTS}
```

```
BEGIN  {MAIN PROGRAM}

CLRSCR;
K := 0;       { initialize }
COUNT := 0;   { initialize }
COUNT_SOL := 2; { include the 2 initial boundary solns }
OPTIMAL := FALSE;
FOR L := 1 TO MAX_NODE DO
  CAND_LIST[L] := -1; { meaningless # to signify empty }

GET_SOLNS;

FIND_MAX(1, MAX1, MAX_POSN1);
FIND_MAX(2, MAX2, MAX_POSN2);
STORE_NODE(0, POINT[MAX_POSN2,1],
           MAX1, POINT[MAX_POSN1,2], MAX2);
      {NODE_NUM,  L1,  U1,  L2,  U2 }
INCUM1 := POINT[MAX_POSN1,1];  { initialize }
INCUM2 := POINT[MAX_POSN1,2];
INCUM := MAX_POSN1;
COMPARE(POINT[MAX_POSN2,1],POINT[MAX_POSN2,2]);
{ find true incumbent }
IF BETTER THEN
  BEGIN
  INCUM1 := POINT[MAX_POSN2,1];
  INCUM2 := POINT[MAX_POSN2,2];
  INCUM := MAX_POSN2;
  END;

IF MAX_POSN1 = MAX_POSN2 THEN OPTIMAL := TRUE;
IF NOT OPTIMAL THEN
  BEGIN

  LOOP:         { Returns to here from goto statements  }
  CHECK_LIST;   { Check candidate list; if not empty,   }
                { select node with smallest number.  If }
                { empty, return EMPTY = TRUE            }

  IF EMPTY THEN RESULTS ELSE  { if empty, current       }
    {incumbent is output and program ends               }
    BEGIN  { continue if not empty }
    GET_NODE;
    IF NOT ((INCUM1 <= NODE[NODE_NUM,2]) AND
    (INCUM2 <= NODE[NODE_NUM,4]))
      THEN   { ideal point of node does not dominate   }
             { incumbent solution                       }
      BEGIN  { compare ideal pt of node with incumbent }
      COMPARE(NODE[NODE_NUM,2],NODE[NODE_NUM,4]);
      IF NOT BETTER THEN GOTO LOOP;
      END;
    K := K + 1;
    HALF := (NODE[NODE_NUM,3] + NODE[NODE_NUM,4])/2;
    MAKE_1STNODE;
    MAKE_2NDNODE;
    GOTO LOOP;
    END;  { NOT EMPTY }
  END;  { NOT OPTIMAL }

END.
```